# PERSIM: AN EVENT-DRIVEN HUMAN ACTIVITY SIMULATOR FOR PERVASIVE SPACES

By

SHANTONU HOSSAIN

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2010

To my parents, Nurun Nahar and Md. Tofazzel Hossain

ACKNOWLEDGMENTS

I would like to convey my earnest gratitude to my advisor Dr. Abdelsalam (Sumi) Helal for his consistent support, guidance, and inspiration throughout my research. I specially thank Jae Woong Lee for his excellent cooperation in designing, developing, and testing the Pervasive Space Simulator-Persim. I also express my gratitude to Dr. Hani Hagras (University of Essex, UK) and Amr Elfaham (German University in Cairo, Egypt) for their wonderful collaboration in developing verification technique for Persim. I thank National Institutes of Health (NIH) for its support towards the Persim project.

I appreciate the cooperation of all the members of Mobile and Pervasive Computing Laboratory in making a great research environment. A special thanks to Chao Chen for his thoughtful suggestions and help.

I thank my husband, Iftekhar Naim for his encouragement and support towards completing my thesis. I also thank all of my friends specially, Dewan Muhammed Ibtesham, Ishtiaq Hossain, Enamul Hoque for being with me through the tough times.

TABLE OF CONTENTS

page

LIST OF TABLES

Table page

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

PERSIM: AN EVENT-DRIVEN HUMAN ACTIVITY SIMULATOR FOR PERVASIVE SPACES

By

Shantonu Hossain

August 2010

Chair: Abdelsalam (Sumi) Helal
Major: Computer Engineering

Access to meaningful collections of sensory data is one of the major impediments in pervasive computing and activity recognition research. Researchers often need data to evaluate the viability of their ideas and algorithms. But obtaining useful sensory data from real world deployments is challenging because of the high cost and significant ground work involved in building actual spaces. Also, the regulatory limitations on human subject use can limit the researcher to execute all possible test scenarios. This situation can be improved by community effort to enable and encourage the sharing of existing inventory of datasets. However, powerful simulation tools and techniques are also needed to satisfy the growing demand of activity data and accelerate research on pervasive and human-centered computing. In this thesis, the Persim project is presented as a solution to these problems by (1) introducing a standard representation of sensory data – Sensory Dataset Description Language (SDDL) for effective sharing of existing datasets among research communities, and (2) contributing a powerful event-driven simulation tool to generate synthetic data for human activities in standardized format. The simulator can capture the physical space in terms of sensors/actuators as well as user behavior (activities) and generate focused simulation

data from the targeted space to achieve a particular research goal. Moreover, the simulator is verified by a fuzzy-based verification technique which assesses the fidelity of the simulated data against the real data collected from smart home deployments.

# CHAPTER 1
# INTRODUCTION

## Motivation

Today, the computing elements are pervading through our environment that brings cyber-physical research as an emergent computing paradigm. Computational and physical elements have become so intertwined with each other that computing is embedded in almost all everyday objects. This has opened a new dimension to various research areas such as assisted living, health care, elder care etc. Researchers are engaged in developing new algorithms and techniques in machine learning and data mining in order to detect activity, learn context, and act autonomously without any human intervention [1]. Thus cyber-computing research involves building and instrumenting a smart space, recruiting participants, and finally collecting data from the space. To obtain data from physical deployments is crucial in order to longitudinally evaluate the accuracy and performance of the developed models and algorithms. But this is very challenging because of the huge cost, significant ground work, lack of access to human subjects, and the time consuming process of acquiring meaningful data from real world settings. Now, the community needs a ‘supportive’ research initiative which will look for alternative and practical approaches to overcome aforementioned challenges and accelerate experiments with the smart space. Realistic simulation is a promising idea to support the rising demand for test data. Simulation also allows a wider community of researchers to engage and collaborate to solve a specific problem. Hence, algorithms and models based on preliminary simulation studies would most likely to be a more robust and help researchers assess their ideas and algorithms quickly and cost-effectively.

## Proposed Solution

In this thesis, we present the Pervasive Space Simulator (Persim) [2]-[4], as a two-fold solution to the critical problem of obtaining human activity data by (1) introducing a standard representation of sensory data for effective sharing of existing datasets among research communities, and (2) contributing a powerful event-driven simulation tool to generate synthetic data for human activities in standardized format. The overview of the project is described in the next section.

### Standardized Representation of Datasets

The scarcity of resources can be mitigated by a community effort to enable and encourage the sharing of existing inventory of datasets. But due to the lack of adequate standard representation of the datasets, it is very difficult (and potentially inaccurate) to utilize other's datasets and conduct advanced research. Therefore, as an early initiative of the Persim project, Sensory Dataset Description Language (SDDL) [5], [6] has been proposed as an XML encoded standard to the research community. SDDL is capable of capturing the pervasive space in terms of its physical elements such as sensors and actuators, as well as the behavior of the residents (activities). It aims to enable proper sharing of datasets across research communities working on areas such as activity recognition and human centric computing. The details of the SDDL structure are described in Chapter 3.

### Simulation of Pervasive Spaces

In order to support the growing demands of data from pervasive spaces, we need to adopt realistic simulation tools and techniques. As a part of the Persim project [2]-[4], we propose the idea of creating a simulated environment of actual pervasive space via modeling activity and generating activity data corresponding to the environment. To

implement the idea, we have developed a simulation engine that can mimic human activities upon specifying the necessary information about the activity. Such simulations are intended to be used in early stage research that can help researchers evaluate their ideas quickly and with reasonable accuracy.

**Persim Project Overview**

From application's point of view, Persim can be divided into two major components: (1) SDDL Converter and (2) Simulation Engine. SDDL Converter can be useful in exploiting the benefits of dataset standardization. It allows the researcher to convert any non-SDDL dataset into SDDL format (which is also the output format of the Simulation Engine) and thus create a level playing field for all. Figure 1-1 shows the overview of the Persim project.

Persim can enable the researcher to create a simulation environment and simulate complex human activities using its web-based interface. It allows one to develop a project in multiple sessions and re-use an already created simulation project by adding, replacing or modifying components to serve new requirements. It also aims to take a dataset in SDDL format and create a simulation project out of it. In this way, it can extend the utility of datasets by adding additional goals or contexts and accelerate the collaborative research among various communities. Moreover, it can also be used as a support tool to take a non-SDDL dataset and convert it to a SDDL format so that datasets can be well-described and easily sharable. Combining all of these use-scenarios, Persim can be used as a leading tool to create a knowledge repository of human activity data.

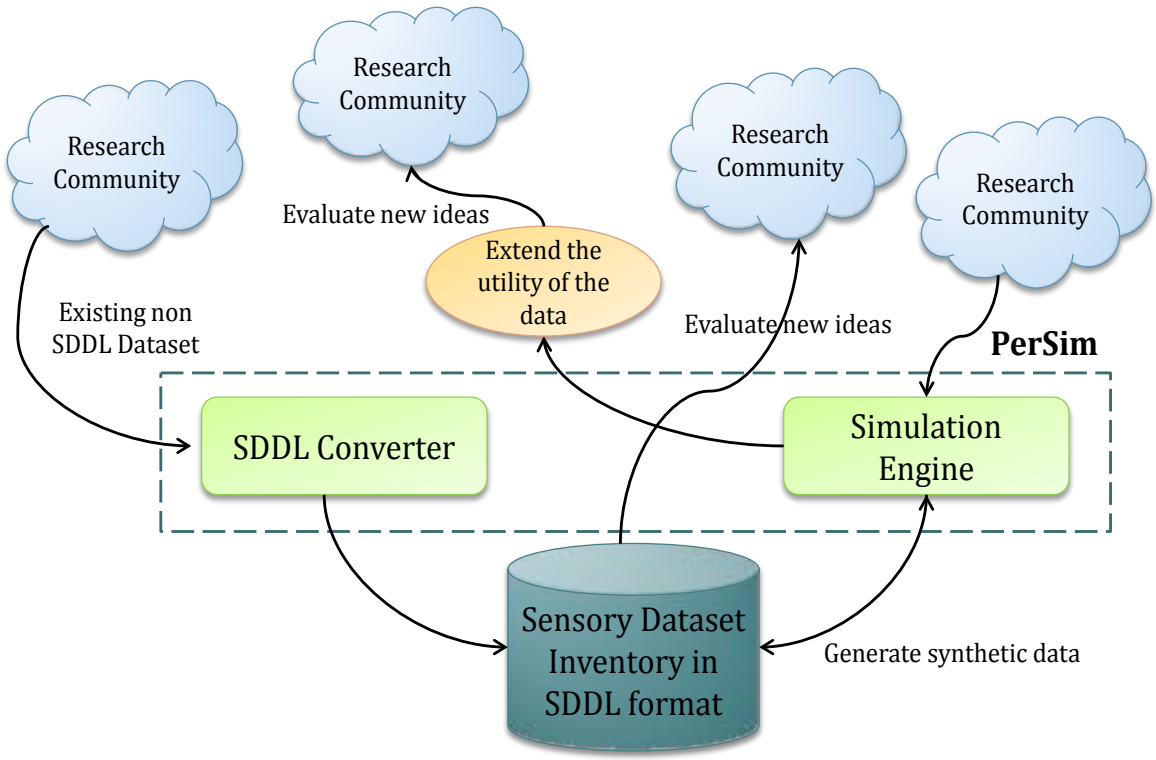


Figure 1-1. High level view of the Persim project

### Organization of the Thesis

The thesis is organized into six chapters. Chapter 1 describes the motivation, problem definition, and the proposed solution. In Chapter 2, the overview on related research work is presented. Chapter 3 discusses Sensory Dataset Description Language (SDDL) in details including goals, development methodology, schema structure, application etc., and Chapter 4 focuses on Pervasive Space Simulator (Persim) and explains the architecture, simulation algorithm, use-scenarios and major features. In Chapter 5, a case study of Persim is presented where a sample problem is described and step-by-step simulation process is explained with screenshots. The thesis is concluded in Chapter 6 by mentioning the future research scopes.

# CHAPTER 2
# RELATED WORK

In this chapter we discuss previous research efforts on standardization of sensor data and simulations of sensor based systems.

## Sensor Data Sharing and Standardization

### SensorML

Sensor Model Language (SensorML) [7] is developed as a part of the Open Geospatial Consortium's (OGC) Sensor Web Enablement (SWE) initiative to provide a standard model and an XML encoding for describing a wide range of sensors and measurement processes. The goal of SensorML is to provide the necessary information to discover, process, and geo-register sensor observations. It defines the dynamic, geometric, and observable characteristics of a sensor. SensorML also assists in on-board processing of data and communication among multiple sensors. The unique feature of this framework is that it models all components such as measurable phenomenon (for example temperature), device that measures/observers a phenomenon (for example temperature sensor), and underlying sensor system (for example a weather station) as discoverable and executable process [7].

### BoxLab Community Effort

BoxLab [8], supported by the National Science Foundation (NSF) and MIT House_n Research Consortium is an initiative for making home activity datasets as a shared resource. The mission of this effort is to include high quality and multi-modal sensor data streams and allow researchers to develop tools and techniques for areas such as context detection and activity detection. This shared knowledge repository can be used to test the viability of ideas and algorithms quickly without engaging

tremendous effort and time to collect data individually from home deployments. Additionally, since the inventory contains most commonly used sensor types (for example RFID, wearable accelerometer, video, audio etc.), researchers will be able to test their algorithms across different scenarios and sensor modalities [8].

## Simulation of Sensor-based Systems

Several simulation concepts and tools have been explored and introduced in simulating sensor-based systems. They all have a different focus and goals.

SENSORIA [9], is a simulator focusing on traffic generation, energy consumption and inherent protocols of WSN. In [10], a detailed simulation model is presented which also focuses on an accurate model for battery, processor power consumption, and network traffic. In [11], Discrete-Event System Specification (DEVS) is proposed to define asynchronous discrete-events occurring in WSN. Eventually, routing and communication becomes non-trivial factors. Unlike Persim, none of the above simulation models focus on human activities performed in pervasive environments.

On the other hand, some approaches such as [12], [13] simulate a specific system at a very low level. For example, TOSSIM [13] simulates the TinyOS operating system in action under some workload (application). However, it is not practical to use TOSSIM to simulate what Persim can simulate easily and without making any assumptions about platforms.

The DiaSim [14] simulator executes pervasive computing applications by creating an emulation layer and developing simulation logic using a programming framework. Under DiaSim, a pervasive computing environment is described in terms of stimulus procedures (any change in the environment that can be consumed by sensors) and simulated services (sensors and actuators) in a specification language called DiaSpec.

It also generates simulation data in raw format. Though Persim has similarity with DiaSim in defining pervasive spaces, the objective of the two simulators is different. Persim aims to simulate human activities performed in pervasive spaces and generate corresponding sensor data in standardized format for further analysis, whereas DiaSim simulates applications such as fire situations and intrusions to identify potential conflicts. Moreover, DiaSim does not attempt to model the entire pervasive space for analysis and examination like Persim does. The validation of DiaSim is performed only from an application's perspective ensuring the pervasive application does what it is intended to do. On the other hand, Persim verifies itself as discussed in Chapter 4 via a fuzzy based verification agent [15] to measure the level of fidelity between simulated system and its real counterpart in terms of their corresponding datasets. It also generates simulated data in SDDL format [5], [6] which makes it suitable for collaborative research.

In [16], an eHomeSimulator is presented which simulates smart environments (called eHomes) as a collection of different services such as heating service and lighting service similar to DiaSim [14]. It focuses on service development, configuration and deployment from software engineering's point of view. It deals with issues like service composition, migration and connecting multiple eHomes. Unlike Persim, it doesn't model human activities and generate activity data.

# CHAPTER 3
# SENSORY DATASET DESCRIPTION LANGUAGE (SDDL)

In this chapter we introduce the XML encoded description language (SDDL), our proposed standard for describing sensor data collected or generated from pervasive system deployments. We organize this chapter by first going over the goals, scope and development methodology of SDDL, then the schema diagram and brief description of each component and finally SDDL features and applications.

## Goals

The primary focus of SDDL project is to initiate a language that can serve as a standard practice for organizing datasets to the research community. It is intended to be an open infrastructure to make dataset repositories easily accessible and sharable. The major goals of SDDL are to:

- Enable the interoperability of datasets and related information.
- Provide a flexible, platform-independent and easy-to-parse interface to archive sensory datasets.
- Facilitate the development of useful tools using this description language to satisfy variety of purposes.
- Focus on the actual research problem without worrying much about dataset organization and sharing.

## Scope

SDDL is intended to describe sensor data observations corresponding to human activities either from real word deployments or from simulated environments. The scope of this description language is to provide the collective information about the pervasive space in terms of available sensors/actuators, activities that can happen inside the space, researcher implicit assumptions, dataset parameters, and labeling. It does not include post processing of sensor data or any physical properties of sensors/actuators.

## Development Methodology

To develop a description language that is intended to be a global representation of sensory data, we realized that it just cannot be done overnight. Unless we repeatedly analyze the existing datasets from different research communities, it is not possible to achieve our desired goal. With this realization, we have adopted an iterative development methodology in designing SDDL.

### Challenges

While developing SDDL we were confronted with several practical challenges - both technical and methodological. We attempted to answer the following questions as we developed the initial version of SDDL:

- How to develop a standard that is not overly generic or not overly specific?
- How can different styles of dataset organization be satisfied by a common format?
- How can the format allow the flexibility to support diverse experiments?
- How to make the language simple to understand as well as easy to parse?

### Approach

Case-study analyses were used as a means for an iterative conjecture-refinement process. While analyzing existing datasets, our objective was to gain an insight of the practice of the researchers while organizing sensor data. Based on the analysis results, we have learned of critical issues related to the characterization of sensor data and the way in which they were collected. Then, we tried to combine different practices in a common design that can serve the major purposes.

The development process was divided into two phases. In the first phase, our focus was to develop an initial structure of SDDL based on a few existing and well-known datasets. In the second phase, previously developed structure was refined and

adapted iteratively to accommodate additional information according to the analysis result and what we have learned from new datasets.

In the first phase shown in Figure 3-1, we have analyzed few datasets and contacted with several researchers (Dr. Diane J. Cook, Washington State University (WSU) regarding CASAS smart home datasets [17] and Dr. Hani Hagras, University of Essex regarding iDorm datasets [18]). Based upon their feedback, we have thoroughly surveyed the datasets and developed the initial version of the schema structure of SDDL v1.0.

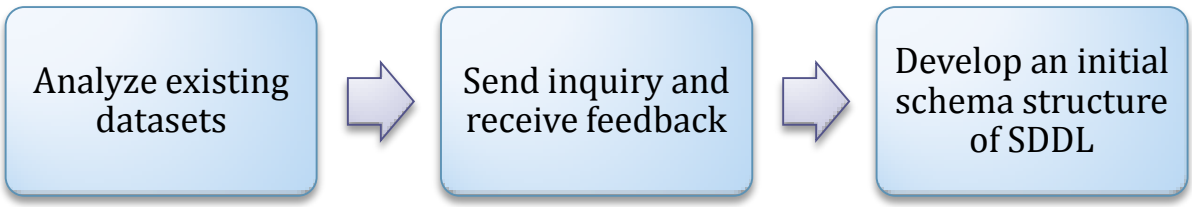


Figure 3-1. SDDL development - phase 1

After phase 1, we have collected few more datasets and started to generalize our initial structure to accommodate new features in an iterative manner. In the second phase, shown in Figure 3-2, we explored following datasets:

- Activity Recognition using Simple and Ubiquitous Sensors, MIT [19], [20].
- MavPad Test-bed of University of Texas at Arlington [21].
- Accurate Activity Recognition in a Home Setting, University of Amsterdam, Netherlands [22].
- Cairo Home Testbed, CASAS Smart Home Project, Washington State University (WSU) [17].

We also communicated with the authorized persons of the datasets whenever we had queries regarding their data collection and organization processes. The details of

the analyses of some datasets are discussed in the next section. After analyzing the datasets we expanded and in some case modified some elements of the initial SDDL as new issues or requirements were identified. This phase is a continuous refinement process that aims to include other research communities and practices to refine the language in order to provide a more universal language support to existing datasets collected from different research projects.

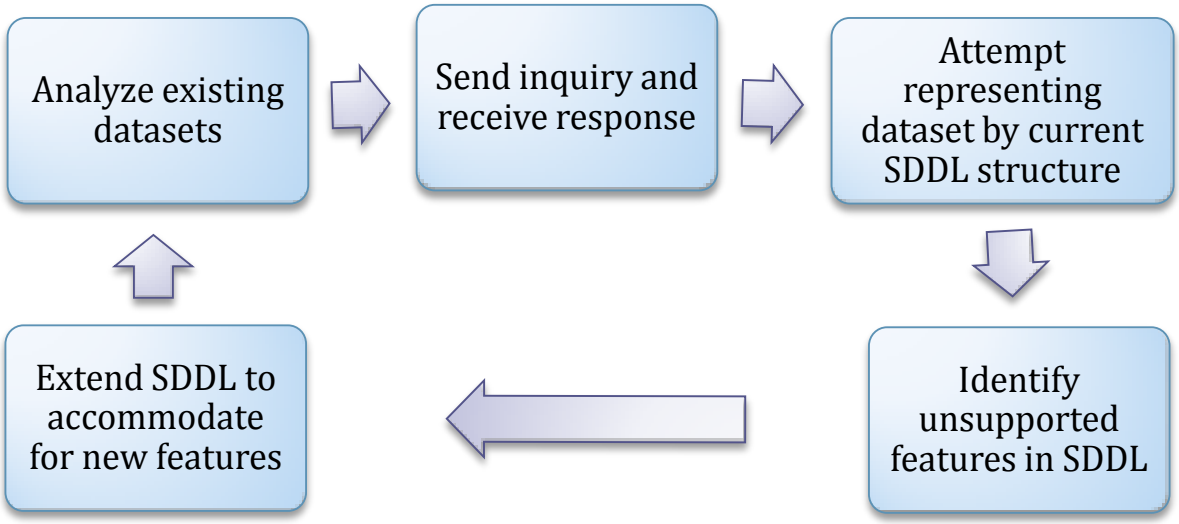


Figure 3-2. SDDL development - phase 2

**Analysis of Existing Datasets**

In the course of analyzing any dataset, first we tried to capture the objective of the experiment and what information is important for the researcher. We found that several datasets tend to store information regarding the description of the experiment itself (e.g. description of the project, space layout, sensor/actuator information, activity information etc.) and the actual data collected from the space in two different files. We realized that it can be a reasonable idea if we could combine these data into a single structure.

**Finding 1**. Categorize sensory datasets into two components, (1) Meta-data (data about the data) and (2) Sensor data. In meta-data, add separate sections for sensor, actuator and activity information.

**Case study 1: CASAS smart home dataset, Washington State University**

The dataset representing sensor events were obtained from the smart three-bedroom apartment located on the Washington State University campus that is part of the CASAS smart home project. The data was collected from students performing five Activities of Daily Living (ADL) – 'Make a phone call', 'Wash hands', 'Cook', 'Eat' and 'Clean' [17]. Each data file corresponds to one participant and one task. It consists of one senor event per line where each sensor event contains date, timestamp, sensor/actuator id, and the sensor value, separated by tab. A snippet of the data (file p02.t2) for the activity - 'Make a phone call' is shown in Figure 3-3.

```
2008-02-29 12:51:58.605865      M13     ON
2008-02-29 12:51:58.605865      M14     OFF
2008-02-29 12:52:01.639008      M14     ON
2008-02-29 12:52:03.542513      M13     OFF
2008-02-29 12:52:04.62222       M14     OFF
2008-02-29 12:52:09.330978      M14     ON
2008-02-29 12:52:09.892831      M13     ON
2008-02-29 12:52:11.646349      M13     OFF
2008-02-29 12:52:13.691857      M15     ON
2008-02-29 12:52:14.149673      M16     ON
2008-02-29 12:52:15.369367      M17     ON
2008-02-29 12:52:17.20986       M14     OFF
2008-02-29 12:52:17.373545      M16     OFF
2008-02-29 12:52:17.48908       M15     OFF
2008-02-29 12:52:19.642209      M17     OFF
2008-02-29 12:52:19.9335        AD1-C   0.0351177
2008-02-29 12:52:22.4941        AD1-C   0.336568
2008-02-29 12:52:37.1656        AD1-C   0.52618
2008-02-29 12:52:38.583396      M17     ON
2008-02-29 12:52:40.16001       AD1-C   0.394342
2008-02-29 12:52:43.969025      M17     OFF
2008-02-29 12:52:44.456067      M17     ON
2008-02-29 12:52:51.250523      M18     ON
2008-02-29 12:52:56.333292      M18     OFF
```

Figure 3-3. A sample of WSU smart apartment's dataset [17]

We observed that the dataset contained single sensor events, that is, it reports only one sensor value per line. But a researcher may want to report all the sensor values at one particular time instance.

**Finding 2**. Add an element 'Sensor Reading Mode' in SDDL schema that denotes the sensor reading mode - either single sensor or multiple sensors.

It is also observed that each data file corresponds to a specific task (e.g. make a phone call, wash hand, cook, eat etc.). Thus, in order to label a group of sensor events with a task, a sub-element is also incorporated to each sensor event element. A sample of WSU dataset representation in SDDL format is shown in Appendix B.

**Case study 2: Intelligent dormitory (iDorm) dataset, University of Essex**

iDorm [18] is a real ambient intelligent environment test bed used for conducting pervasive computing research at Essex University, UK. The dataset was collected several times from three users over a period of three days. It was recorded over different months of the year in order to reflect the seasonal variations in the dataset. Each sensor event in the dataset is separated by a line feed and consists of the semicolon separated values of sensor id's and readings of all eight sensors and ten actuators. A snippet of the dataset named 'dataset_user1_June2004' is shown in Figure 3-4.

Here, first observation is that the dataset corresponds to multiple sensor/actuator events and this issue is already handled by the language via Sensor Reading Mode. Secondly, the dataset includes a labeling mechanism by which sensor readings (or files) are labeled with a particular context (e.g. month of the year). In this way, the sensor values can reflect the seasonal variations in the environment. Therefore, we realized that such feature is very important for further research and allows the data collectors to

describe contexts in which the data was collected, that only he/she knows of. We added the element ‘Active Context’ as another metadata in SDDL.

InternalLightLevel:10;ExternalLightLevel:72;InternalTemperature:24.47;ExternalTemperature:13.5;ChairPressure:0;BedPressure:1;Hour:9.183333333333334;ACTION_Light1_value:0;ACTION_Light2_value:45;ACTION_Light3_value:0;ACTION_Light4_value:0;ACTION_Blind_state:0;ACTION_BedLight_state:0;ACTION_DeskLight_state:0;ACTION_Heater_state:0;ACTION_MSWord_state:0;ACTION_MSMediaPlayer_state:0

InternalLightLevel:60;ExternalLightLevel:73;InternalTemperature:24.5;ExternalTemperature:13.5;ChairPressure:0;BedPressure:1;Hour:9.25;ACTION_Light1_value:0;ACTION_Light2_value:45;ACTION_Light3_value:0;ACTION_Light4_value:0;ACTION_Blind_state:0;ACTION_BedLight_state:0;ACTION_DeskLight_state:0;ACTION_Heater_state:0;ACTION_MSWord_state:0;ACTION_MSMediaPlayer_state:0

InternalLightLevel:77;ExternalLightLevel:73;InternalTemperature:24.5;ExternalTemperature:13.5;ChairPressure:0;BedPressure:0;Hour:9.266666666666667;ACTION_Light1_value:0;ACTION_Light2_value:2;ACTION_Light3_value:0;ACTION_Light4_value:0;ACTION_Blind_state:0;ACTION_BedLight_state:0;ACTION_DeskLight_state:0;ACTION_Heater_state:0;ACTION_MSWord_state:0;ACTION_MSMediaPlayer_state:0

InternalLightLevel:77;ExternalLightLevel:73;InternalTemperature:24.54;ExternalTemperature:13.5;ChairPressure:1;BedPressure:0;Hour:9.35;ACTION_Light1_value:0;ACTION_Light2_value:2;ACTION_Light3_value:0;ACTION_Light4_value:0;ACTION_Blind_state:0;ACTION_BedLight_state:0;ACTION_DeskLight_state:0;ACTION_Heater_state:0;ACTION_MSWord_state:0;ACTION_MSMediaPlayer_state:0

InternalLightLevel:101;ExternalLightLevel:77;InternalTemperature:24.54;ExternalTemperature:13.5;ChairPressure:1;BedPressure:0;Hour:9.433333333333334;ACTION_Light1_value:0;ACTION_Light2_value:2;ACTION_Light3_value:0;ACTION_Light4_value:0;ACTION_Blind_state:0;ACTION_BedLight_state:0;ACTION_DeskLight_state:0;ACTION_Heater_state:0;ACTION_MSWord_state:0;ACTION_MSMediaPlayer_state:0

InternalLightLevel:81;ExternalLightLevel:73;InternalTemperature:24.5;ExternalTemperature:13.5;ChairPressure:0;BedPressure:0;Hour:9.533333333333333;ACTION_Light1_value:0;ACTION_Light2_value:2;ACTION_Light3_value:0;ACTION_Light4_value:0;ACTION_Blind_state:0;ACTION_BedLight_state:0;ACTION_DeskLight_state:0;ACTION_Heater_state:0;ACTION_MSWord_state:0;ACTION_MSMediaPlayer_state:0

Figure 3-4. A sample of iDorm dataset [18]

It is also observed that datasets vary in terms of number of parameters and how they are separated. Without this knowledge, the datasets cannot be understood or parsed by others.

**Finding 3**. Allow SDDL to capture the listing of the parameters and corresponding separators in such a way that any number of parameters and any type of separators can be supported.

**Case study 3: Activity recognition in home setting using simple and ubiquitous sensors, Massachusetts Institute of Technology**

The objective of this experiment was to recognize Activity of Daily Living (ADL) carried by a single person in a home setting. In this work, the key approach was to decompose human activities as a sequence of binary sensor activations. For this, sensors were installed in everyday objects like refrigerator, drawer, remote control, etc., so that any movement or opening/closing (activation/deactivation) action can be detected and attributed to the activity being performed while using those objects [19], [20]. Figure 3-5 shows a snippet of data file 'activities_data.csv' of 'subject 1'.

The dataset corresponds to data about human activity performed in two single-person apartments for two weeks. In each apartment, between 77 and 84 sensor data collection board equipped with reed switch sensors were installed. In this experiment, Supervised Learning approach was followed and Naïve Bayesian Classifier was used to detect activities [19], [20].

The dataset consists of three types of data in three separate files. These are:

- **List of activities**. This file consists of all predefined list of activities corresponding to the experiment. Each activity is defined by comma separated values of heading, category, sub-category, and code.
- **List of sensors**. This file contains all the sensors installed in the apartment. Each sensor is defined by comma separated values of sensor ID, location, and object (to which the sensor is attached).
- **Activity data**. This is the actual data file which contains the information about the activity which was being performed and the sensors which were active during that time. Each entry of the dataset corresponds to an activity which is defined by comma separated values of activity label, start time, end time, sensor ID, sensor object, sensor activation time, and sensor deactivation time.

At this point we had a few major observations. First, the dataset has pre-defined sensor labeling and activity labeling. Second, the actual activity-data is also labeled with

activity and sensor information corresponding to that activity. Third, different activities are defined by different subsets of sensors. For example, in Figure 3-5, 'Toileting' activity is defined by sensors which are attached to the Sink faucet – hot and light switch. On the other hand, 'Preparing Lunch' activity is defined by sensors that are attached to the Door, Freezer, Toaster, Cabinet, and Drawer. Fourth, same activity can also be defined by different subsets of sensors. For example, same activity 'Preparing Lunch' can have different sensors at different time.

```
Bathing,4/1/2003,20:41:35,21:32:50
100,68,81,101,93,137,93,58,57,67,93,58,68,88,57,67,100,68,67,76
Toilet Flush,Sink faucet - hot,Closet,Light switch,Shower faucet,Freezer,Shower faucet,Medicine
cabinet,Medicine cabinet,Cabinet,Shower faucet,Medicine cabinet,Sink faucet - hot,Sink faucet -
cold,Medicine cabinet,Cabinet,Toilet Flush,Sink faucet - hot,Cabinet,Lamp
20:51:52,20:51:58,20:53:36,20:53:49,20:53:52,20:58:22,20:58:43,21:5:23,21:5:46,21:5:47,21:18:34,21:1
8:55,21:19:41,21:20:4,21:20:38,21:20:39,21:21:13,21:21:16,21:21:37,21:22:8
21:5:20,20:52:5,20:53:43,21:21:43,20:58:42,20:58:32,21:6:9,21:5:45,21:18:55,21:5:49,21:18:35,21:20:3
7,21:20:5,21:20:34,21:21:41,21:20:42,23:10:23,21:21:23,21:21:38,23:11:8

Toileting,4/1/2003,17:30:36,17:46:41
100,68
Toilet Flush,Sink faucet - hot
17:39:37,17:39:46
18:10:57,17:39:52

Toileting,4/1/2003,18:4:43,18:18:2
68,107
Sink faucet - hot,Light switch
18:11:2,18:12:28
18:11:13,21:21:53

Going out to work,4/1/2003,12:11:26,12:15:12
81,139,140
Closet,Jewelry box,Door
12:12:29,12:13:27,12:13:45
12:13:0,12:13:35,12:13:48

Preparing lunch,4/1/2003,11:21:17,11:38:22
140,137,131,53,84,131
Door,Freezer,Toaster,Cabinet,Drawer,Toaster
11:23:4,11:23:55,11:24:8,11:34:59,11:35:4,11:35:12
11:23:7,11:24:3,11:24:14,11:35:1,11:35:7,11:35:22
```

Figure 3-5. A snippet of activity recognition dataset, MIT [19], [20]

Based on the above observations, we concluded that the SDDL version we had at the time of this analysis had all the necessary elements (discussed in the next section) to accommodate this dataset. We can incorporate the list of sensors in the Sensor section, list of activities in the Activity section and activity data in Sensor data section. Also, if expressed in SDDL, all three separate files mentioned above can be merged to one SDDL file which can save a good amount of space.

**Case study 4: MavPad test bed, University of Texas at Arlington**

MavPad is a test bed of MavHome (Managing and Adaptive Versatile Home) project of University of Texas at Arlington. It is an on-campus apartment for a single resident. The objective is to develop an intelligent environment by automating the interactions of the inhabitant and dynamically adapting the concepts that change over time. In this project, Hierarchical Hidden Markov Model (HHMM) is constructed using the output of data-mining algorithm and Episode Discovery were used in solving automation problems in the intelligent environment domain [21].

To conduct the experiment, a sensor network is developed in the whole apartment to percept through light, smoke, temperature, humidity, motion, etc. The Argus network system is used which is comprised of a master board that interfaces to the computer via a serial interface and connects up to 100 slave boards that host up to 64 sensors each. The dataset of MavPad represents sensor events collected during 2005 while a single resident was living in the apartment [21].

According to our observation, the dataset contains raw data having parameters Date, Time, Zone, Device ID, State, Level/Value, Source, and Info. A part of the dataset from file '1-3-2005-RAW.data' is shown Figure 3-6. It also has a separate file for mapping different sensors/actuators to ID, type, location, etc. We also observed that

each line in the data set corresponds to single sensor event, that is, only one sensor is listed against a specific timestamp. On the other hand, each sensor event is identified by sensor location (zone), device ID, sensor's state (either active or not active) and sensor's value.

**Finding 4.** Add 'Location' section in metadata to associate sensors/actuators with their physical position or orientation in the target space.

| mark | zone | number | state | level | source | info |
|---|---|---|---|---|---|---|
| 2005-01-03 09:47:30 | i | 5 | 1 | 100 | X10 | inhabitant |
| 2005-01-03 09:56:17 | i | 5 | 0 | 0 | X10 | inhabitant |
| 2005-01-03 13:04:45 | a | 1 | 1 | 100 | X10 | inhabitant |
| 2005-01-03 13:05:37 | i | 3 | 1 | 100 | X10 | inhabitant |
| 2005-01-03 13:06:11 | c | 4 | 1 | 100 | X10 | inhabitant |
| 2005-01-03 13:06:22 | c | 4 | 0 | 0 | X10 | inhabitant |
| 2005-01-03 13:16:32 | S | 1 | 1 | 10 | ArgusMS | inhabitant |
| 2005-01-03 13:16:33 | S | 2 | 1 | 152 | ArgusMS | inhabitant |
| 2005-01-03 13:16:33 | S | 3 | 1 | 13 | ArgusMS | inhabitant |
| 2005-01-03 13:16:33 | S | 5 | 1 | 11 | ArgusMS | inhabitant |
| 2005-01-03 13:16:33 | S | 6 | 1 | 159 | ArgusMS | inhabitant |
| 2005-01-03 13:16:33 | S | 7 | 1 | 111 | ArgusMS | inhabitant |
| 2005-01-03 13:16:33 | S | 8 | 1 | 12 | ArgusMS | inhabitant |
| 2005-01-03 13:16:33 | S | 9 | 1 | 158 | ArgusMS | inhabitant |
| 2005-01-03 13:16:33 | S | 13 | 1 | 56 | ArgusMS | inhabitant |
| 2005-01-03 13:16:33 | S | 14 | 1 | 67 | ArgusMS | inhabitant |

Figure 3-6. A snippet of MavPad testbed, University of Texas at Arlington [21]

## Overview of SDDL Schema Structure

### Terms and Notations

In this thesis, the following terms and notations are used to describe SDDL schema elements:

- Element name starts with an upper-case letter. Any attribute name starts with a lower-case letter.
- Element and attribute names use an underscore ("_") to separate multiple words in order to improve readability (For example, <Sensor_Info>).
- Elements can be simple or complex.
- A simple schema element is the one that does not have any attributes.

- A complex schema element is the one that can have multiple simple or complex sub elements. It can also have several attributes. Sub-elements are described in orderly fashion after the parent element.

- Not all schema elements are mandatory in SDDL. In the schema diagram, the mandatory elements are shown in solid rectangle where as optional elements are shown in dashed rectangle.

**Schema Elements and Attributes**

SDDL is designed as a hierarchical collection of elements and their attributes. One element can contain one or more sub-elements. Each element can also have one or more attributes. Now we shall explain all the schema elements with some details. The complete schema definition of SDDL is provided in Appendix A.

**<Sensory_Dataset>.** This is the root element of the SDDL schema that captures both metadata and sensor data of a dataset. Figure 3-7 shows the schema diagram of this element. It has the following attributes:

- version. The version number of the SDDL instance.
- id. The identification number of the dataset.
- name. The name of the dataset.
- date_from. The start date and time of the sensor data generation.
- date_to. The end date and time of the sensor data generation.

**<Contact_Info>.** An element that briefly describes the contact information (e.g. name, phone, email, etc.) of the authorized person of the dataset. The schema diagram is shown in Figure 3-8. This element has the following attributes:

- name. The name of the authorized person of the dataset.
- role. The role of the authorized person such as owner, lab manager, etc.
- organization. The name of the corresponding organization.
- phone. The contact number of the authorized person of the dataset.
- email. The email address of the authorized person.

**<History>.** This element contains all necessary details of the pervasive space where data is generated. The schema diagram for this element is shown in Figure 3-8.

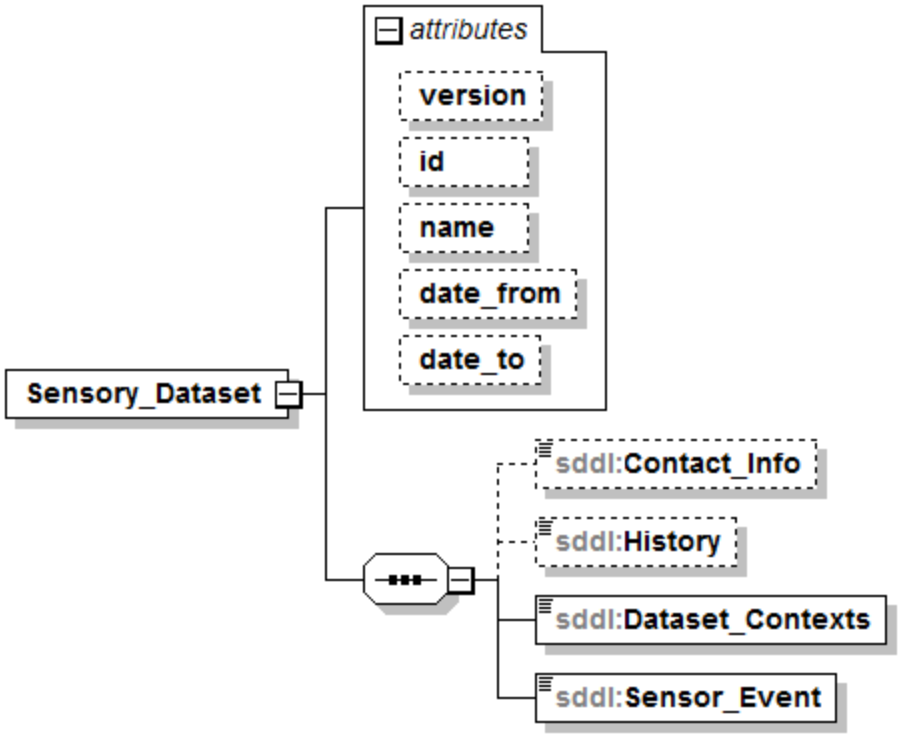


Figure 3-7. Schema diagram of <Sensory_Dataset> element in SDDL

It includes three simple sub-elements - <Project_Description>, <Dataset_Description>, and <Space_Layout> and one complex element - <Project_Specification>. <Project_Description> and <Dataset_Description> briefly describe the general information about the project and dataset. <Space_Layout> represents information about the area and the layout of the target space.

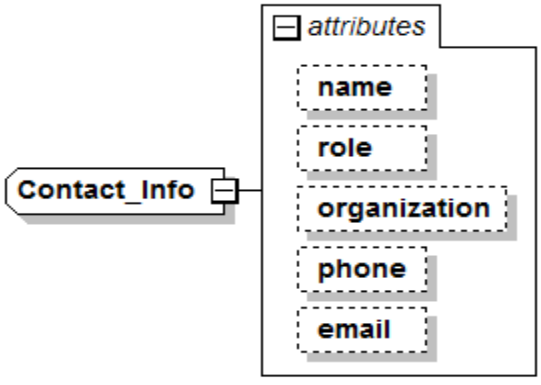


Figure 3-8. The schema diagram of <Contact_Info> element in SDDL

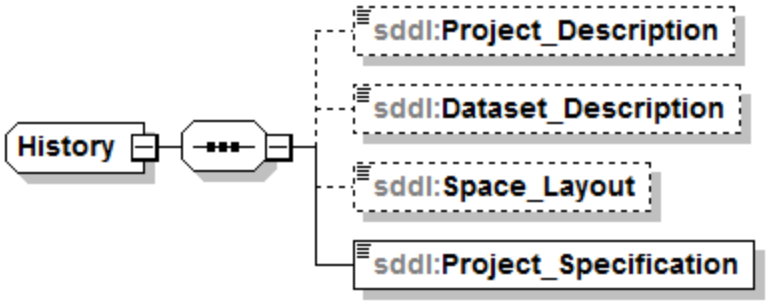


Figure 3-9. The schema diagram of <History> element in SDDL

**<Project_Specification>.** It provides the necessary information about deployed sensors and actuators, locations in the space, defined activities and active contexts related to the dataset etc. It includes six complex sub-elements described below. The schema diagram is shown in Figure 3-10.

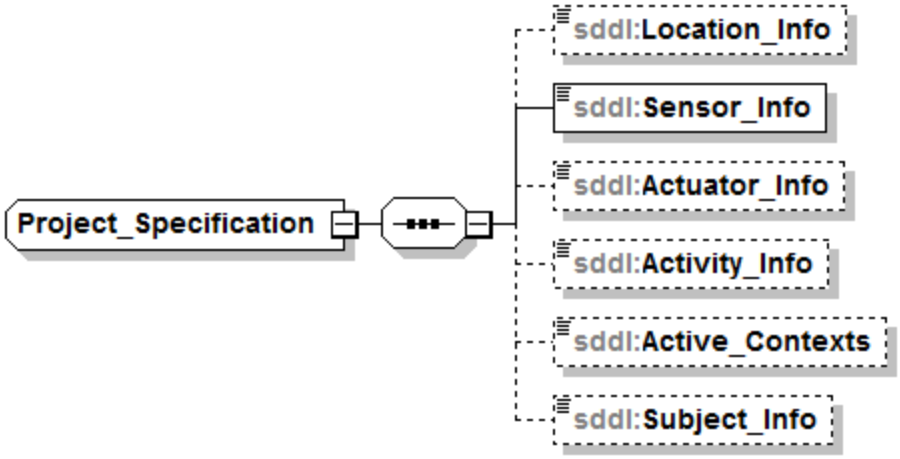


Figure 3-10. The schema diagram of <Project_Specification> element in SDDL

**<Location_Info>.** It contains information about area in the space where sensor or actuators are physically located. Count attribute denotes the number of such areas. The schema diagram of the element is shown in Figure 3-11. It can contain one or more sub-elements <Location> which represents individual locations. <Location> element has the following attributes:

- id. The identifier of the particular area in the space.
- name. The name of the location such as bedroom, bathroom etc.

- zone. The name of the zone if an area is divided into zones.

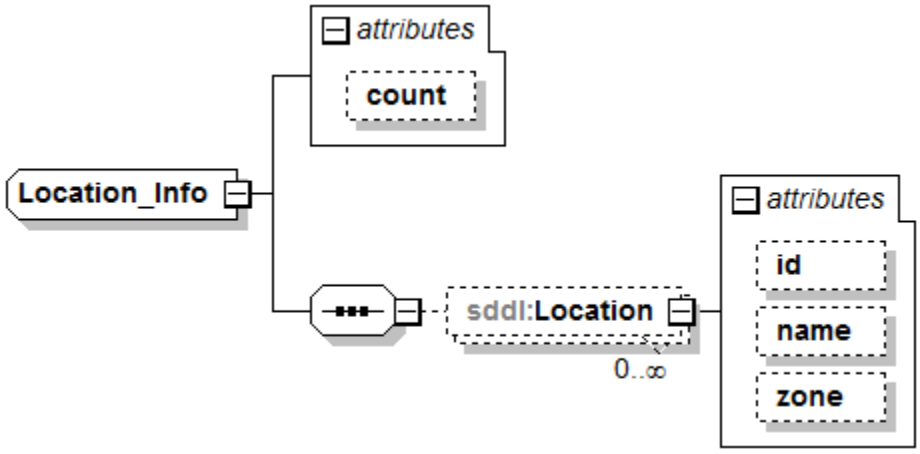


Figure 3-11. The schema diagram of <Location_Info> element in SDDL

**<Sensor_Info>.** This element contains the relevant information about all the sensors that are available in the space. It has count attribute which denotes the number of sensors. The schema diagram is shown in Figure 3-12. Here, <Sensor_Info> can contain one or more complex element <Sensor> which describes the characteristics of a particular sensor. It has the following attributes:

- id. The identifier of the particular sensor.
- name. The name of the sensor.
- type. The functional type of the sensor such as motion sensor, light sensor, etc.
- location_id. The location identifier of the sensor mentioned in <Location> element.
- unit. The measurement unit of the sensor value in textual form.
- min_value. The minimum output value range of the sensor.
- max_value. The maximum output value range of the sensor.

**<Actuator_Info>.** This element contains the relevant information about all the actuators that are available in the space. It has count attribute which denotes the number of actuators. The schema diagram is shown in Figure 3-13. Here, <Actuator_Info> can contain one or more complex element <Actuator> which describes the characteristics of a particular actuator. It has the following attributes:

- id. The identifier of the particular actuator.
- name. The name of the actuator.
- type. The functional type of the actuator such as servo, electric motor, etc.
- location_id. The location id of the actuator mentioned in <Location> element.
- unit. The measurement unit of the actuator value in textual form.
- min_value. The minimum output value range of the actuator.
- max_value. The maximum output value range of the actuator.

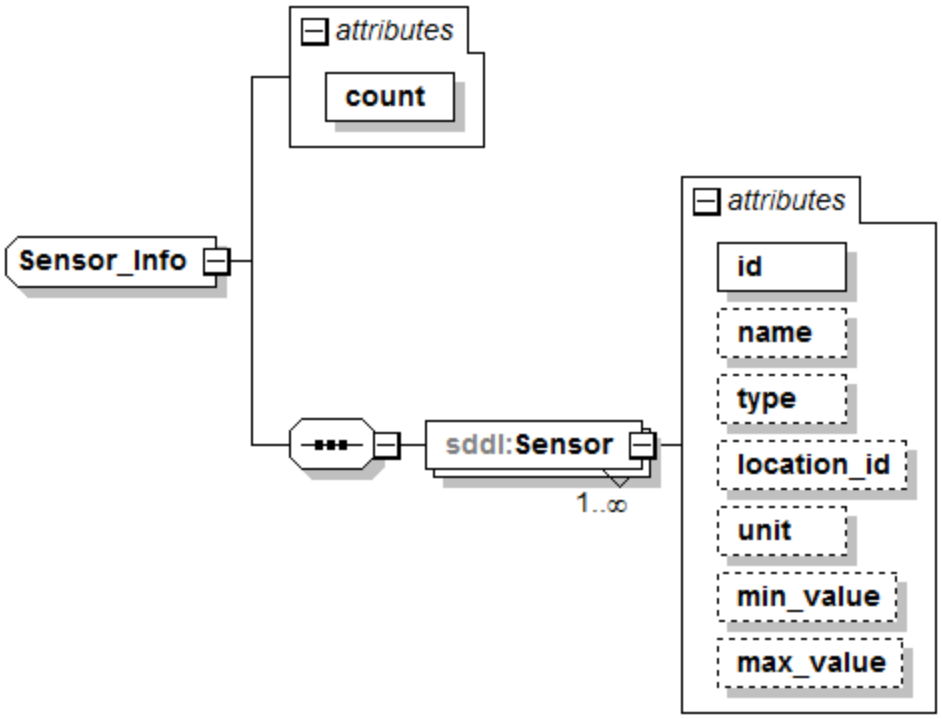


Figure 3-12. The schema diagram of <Sensor_Info> element in SDDL

**<Activity_Info>.** This element denotes the activities or tasks that can happen inside the space. The number of activities is stored in 'count' attribute. Figure 3-14 shows the schema diagram of the element. The <Activity> sub-element describes information about a particular activity and has the following attributes:

- id. The identifier of the activity.
- name. The name of the activity.
- category. The category of activities such as housekeeping, food preparation etc.
- sub-category. The sub-category of activities, if any.

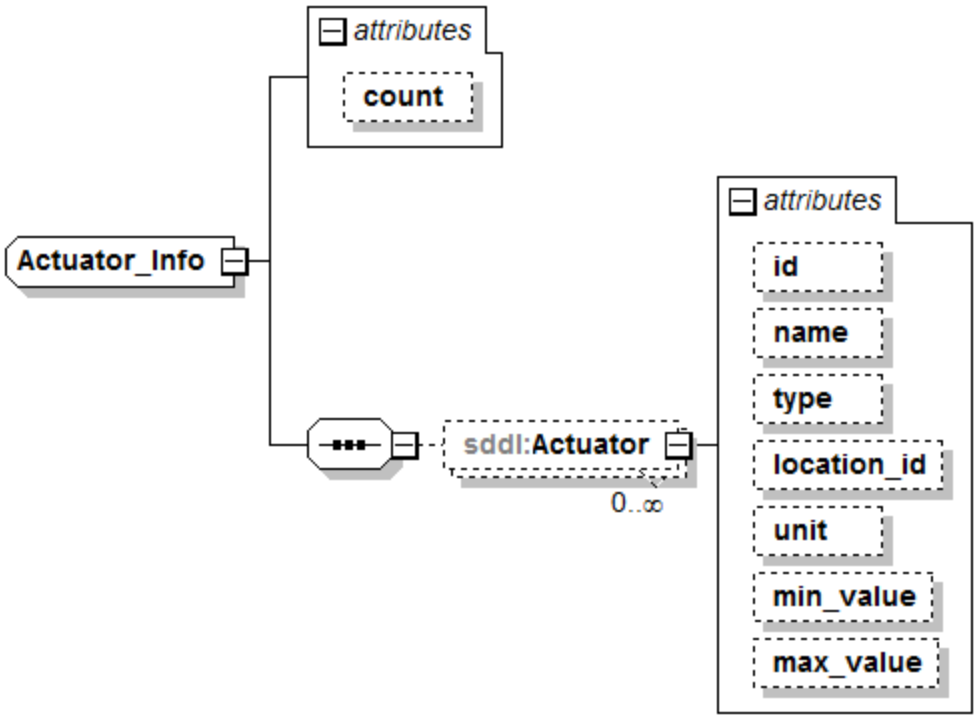


Figure 3-13. The schema diagram of <Actuator_Info> element in SDDL

**<Active_Contexts>.** The element denotes the contexts that are involved while collecting data from the space. For example, 'age of the participant', 'month of the year', can be example of such context. The number of contexts is stored in 'count' attribute.

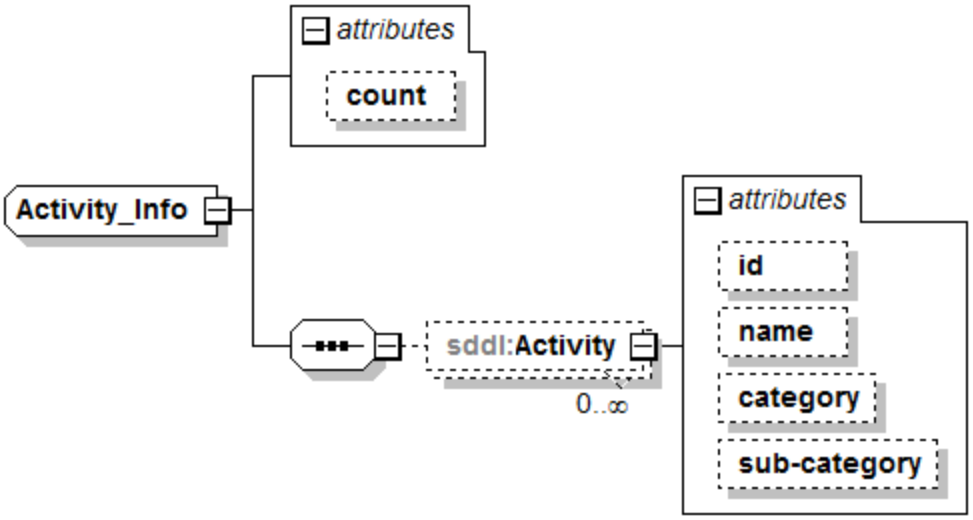


Figure 3-14. The schema diagram of <Activity_Info> element in SDDL

The schema diagram of this element is shown in Figure 3-15. It consists of sub-element <Active_Context> that has the following attributes:

- id. The identifier of the context.
- name. The name of the context.
- type. The type or category of the context.

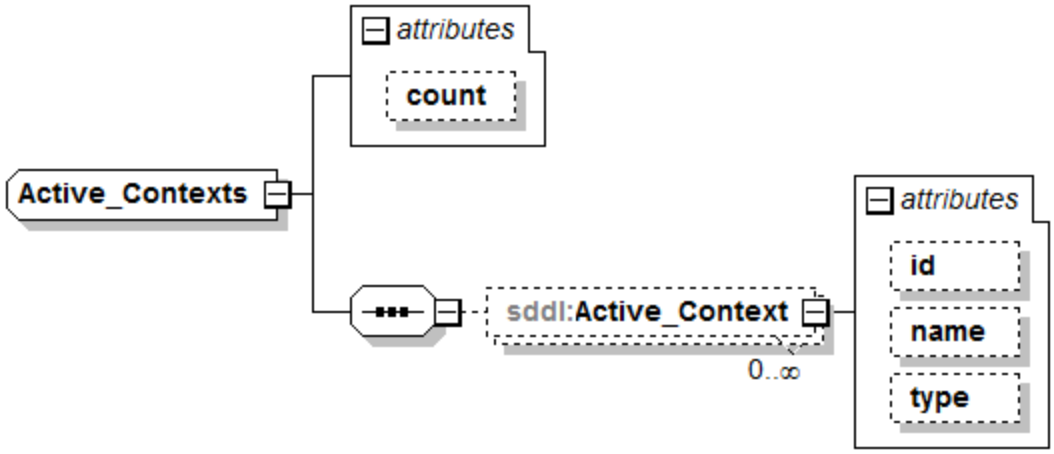


Figure 3-15. The schema diagram of <Active_Contexts> element in SDDL

**<Subject_Info>.** This element contains meta-data about the participants of the experiment. The number of subjects is stored in 'count' attribute. The schema diagram of the element is shown in Figure 3-16. It contains <Subject> sub-element that has the following attributes:

- id. The identifier of the subject.
- name. The name of the subject.
- age. The age of the subject.
- ethnicity. The ethnicity of the subject.
- gender. The gender of the subject.
- disease/disability condition. The disease or disability condition such as obesity and diabetes.

**<Dataset_Contexts>.** The element contains the implicit information about the organization and representation of the dataset such as parameters and the separator of

the parameter. The schema diagram of the element is shown in Figure 3-17. It has three sub-elements - <Parameters>, <Separators>, and <Sensor_Event_Type>.

The <Parameters> can have several <Parameter> which denotes the information that each sensor event captures. For example, parameters can be timestamp, sensorID, sensorValue etc. The number of parameters is represented by the 'count' attribute. The <Parameter> sub-element has the following attributes:

- name. The name of the parameter that is present in the sensor data.
- index. The index number where the parameter is placed in the sensor data.

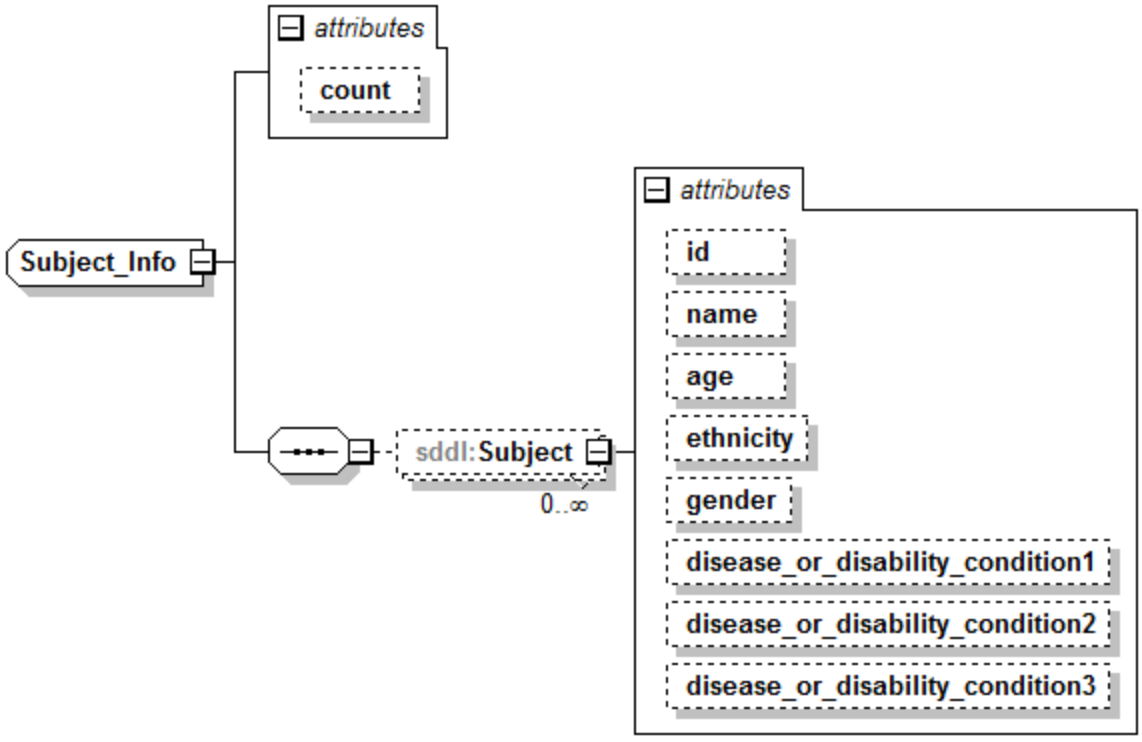


Figure 3-16. The schema diagram of <Subject_Info> element in SDDL

<Seperators> sub element refers to the various separators such as comma, semicolon, space, etc., used in representing the particular sensor data instance. It has the following attributes:

- parameter_separator. The separator between parameters in the dataset.
- line_separator. The separator between two lines/data instances in the dataset.

<Sensor_Event_Type> refers to the mode of sensor data reporting. It is restricted to two values - single_sensor_per_event (i.e. only one sensor reading is recorded per line) and multiple_sensors_per_event. (i.e. multiple sensor readings are recorded in one line).

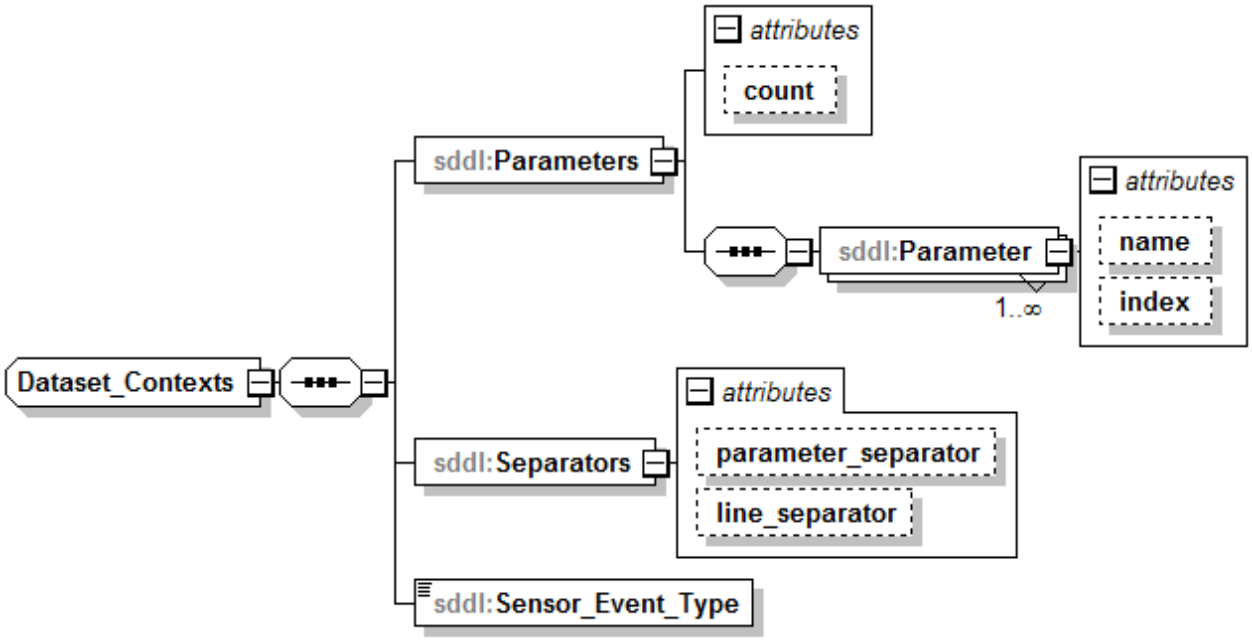


Figure 3-17. The schema diagram of <Dataset_Contexts> element in SDDL

**<Sensor_Event>.** The element represents the actual sensor data collected from the space. It also contains labeling information such as 'performed activity' and 'active context'. The schema diagram of the element is shown in Figure 3-18.

The element contains <Event> sub element which denote a collection of sensor data that may correspond to an activity under certain context. There can be multiple instances of <Event> element for representing sensor data corresponding to different activities. It has the following attributes:

- data. The numeric sensor data collected from space.
- activity_performed. The activity that was being performed during data collection.
- active_context. The activity context under which the data was collected.

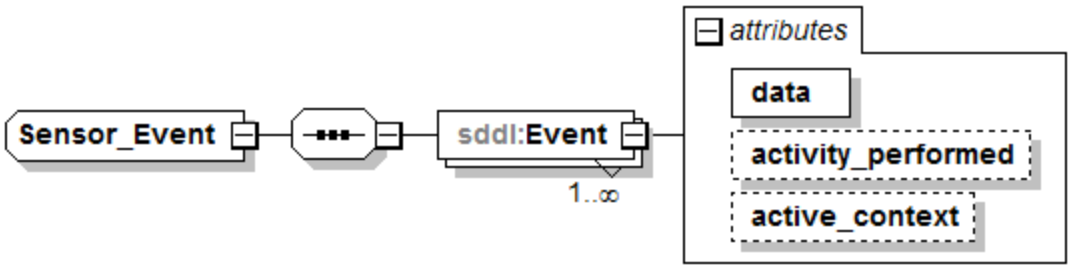


Figure 3-18. The schema diagram of <Sensor_Event> element in SDDL

**SDDL Vs SensorML**

Both Sensory Dataset Description Language (SDDL) [5], [6] and Sensor Model Language (SensorML) [7] are initiatives towards describing sensor based systems but they differ in the focus and field of applications. SensorML provides a framework that defines the dynamic, geometric and observational characteristics of sensors and sensor based systems. The major focus of SensorML is to make the phenomenon (observation) and the physical device that observers the phenomenon (sensor), discoverable and executable. SensorML models any sensor, observable phenomenon or sensor system as a process having input, output, parameters and metadata. On the other hand, SDDL focuses on dataset representation of sensor based systems by capturing the system in terms of sensors and actuators, dataset contexts, parameters, and separators etc. It also contains sensor data collected from the pervasive spaces. The concept of SDDL is much simpler and easily understandable compared to complex process based concept of SensorML. The goal of SDDL is to serve as a common syntax of representing human activity data in order to leverage the community to analyze, reuse and extend the data for advanced research. In this context, it is not useful to model all the elements of the space into a complicated structure like SensorML and include geometric characteristics, measurement process etc. But for the remote

sensor systems that require features such as sensor discovery, on board processing, and geo-register sensor observations, SensorML can be very useful. Thus there is no conflict of interests between SDDL and SensorML, rather they both can work in parallel to serve individual purposes.

## Applications

### Organize Datasets in a Structured Way

SDDL can serve as a guideline to organize sensory datasets in a common skeleton culminated from various deployments. Its simple but powerful infrastructure enables a clean separation between meta-data and sensor data. It can be utilized in archiving numerous sensor data and parsing them with minimum effort. It also eliminates the redundant information and stores all information in a compact way.

### Share of Other's Dataset Effectively

SDDL provides a self-explanatory presentation of data so that any person who is not familiar with the dataset can get the insight of the data very quickly and efficiently. Thanks to XML, each schema element and its attributes are easily understandable. By following the 'grammar' of representing dataset, researchers can re-use and share other's dataset more effectively [6].

### Create Dataset Inventory

Many research areas such as Speech Recognition and Face Detection have their own repository of datasets which have facilitated the research. BoxLab community effort in making home activity data as shared resource has been well-appreciated [8]. But unless there is an agreed-upon dataset standard, it will be difficult to achieve all the benefits of this type of initiative. Thus SDDL can act as a catalyst in creating and supporting an inventory of datasets and make it widely used.

**Facilitate the Development of Useful Tools**

As an open standard of dataset representation, SDDL can promote the development of various tools that will be useful in the research of activity recognition or human centric computing. This can bring the methodological improvement in analyzing, simulating and verifying human activity data and conduct advanced research.

# CHAPTER 4
# PERVASIVE SPACE SIMUALTOR (PERSIM)

In this chapter, we describe our idea of pervasive space simulation - how we modeled the space in terms of physical devices and user-behavior. We also describe the simulation algorithm which captures the dynamic nature of the pervasive space and generates discrete events to simulate desired human activities.

## Limitations of Actual Space Deployment

To build an actual pervasive space, deploy devices, and execute desired experiments, there are several practical limitations. Few major issues are explained below:

### High Cost of Building Pervasive Spaces

To build a pervasive space with correct design and planning is very expensive. It requires significant ground work and involvement of area specialists to create the smart home with necessary devices and appliances. Not everybody has a large budget to build such a space. Moreover, integrating and connecting various heterogeneous devices and collecting data from the system are also very challenging [15].

### Difficulty to Recruit Human Subjects

Human subjects are not always easy to find and recruit. Even subjects are available, they cannot be used to perform all the activities under all possible conditions or contexts that a researcher wishes to consider. To ensure safety and prevent against abuse and exploitation, many governmental agencies and institutional review board restrict the length of time human subjects can be used in any research study [15].

**Significant Time to Generate Data**

It is usually very time consuming to generate the adequate data for a meaningful collection of events for all possible test scenarios. Depending on the goals of the experiment, it can take weeks to generate activity data from a real deployment.

**Difficulty to Modify the Physical Space**

A physical space is not scalable in the sense that it cannot be extended easily by adding/replacing sensors or changing the layout. Thus all the experiments that are conducted have strong coupling with the hardware infrastructure of the space. Also, any alteration to the experimental setup is time consuming and associated with financial cost [16]. Additionally, once data is collected from the space, it is impossible to modify the setup without repeating the experiment again.

**Inability to Reproduce Experimental Data**

Sometimes, researcher may find the collected amount of data is not adequate for learning his/her activity model and thereby more data corresponding to the same activity under same experimental setup is needed. But, it is not possible to reproduce the similar activity data from the space without running the experiment again.

## Challenges

In order to design a simulator that is free from the limitations of actual space deployment and capable of generating human activities, we have faced a number of challenges. Few of those are:

- How to define an actual space in terms of its intrinsic elements such as area, sensors, actuators and extrinsic elements or events like ‘a person is cooking’, ‘room temperature is changed from 60F to 62F’etc.?
- How to support the large set of diverse and heterogeneous elements of the space and extend the utility of the simulator over time?

- How to define the semantic of an activity in terms of sensors?
- What parameters are essential for the simulation model to generate activity events?
- How to represent the simulated dataset to the world in a way to foster collaborative research?

**Persim Architecture**

We exploited the Model-View-Controller (MVC) [23], a simple yet powerful architectural pattern to develop Persim. According to this pattern, the architecture is divided into three components - model, view and controller so that there is a clean separation among the components and one can be modified without affecting the others. This feature allows Persim to be extendable without much effort. Figure 4-1 represents the overview of the Persim architecture.

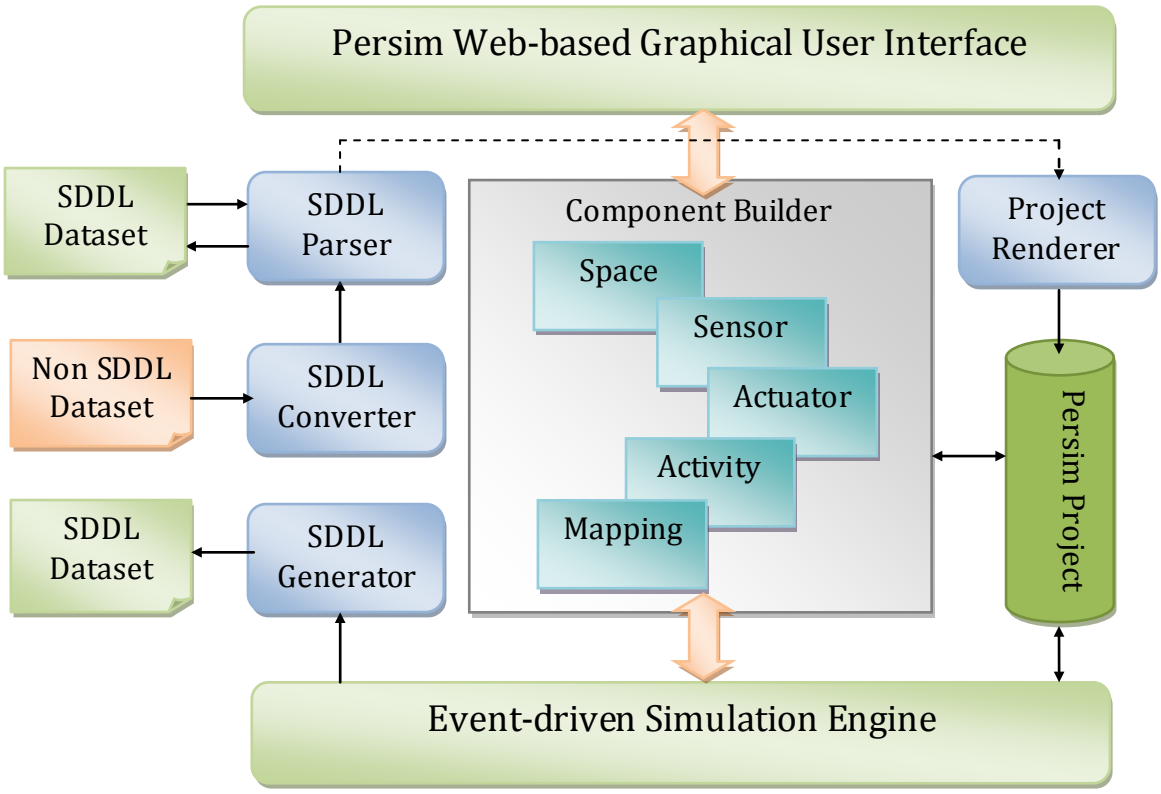


Figure 4-1. Persim architecture

In Persim architecture, the 'model' represents the data structures that store all the required information of the simulation environment such as Space, Sensor, and Activity. The 'view' presents the front end of the simulator that contains graphical representation of the simulated environment. The 'controller' communicates with the front end and processes user interaction and updates the view as the model changes.

In Persim, user interacts with the simulator through a web-based graphical user interface (GUI). The simulation environment is constructed by the components such as space, sensor, activity, mapping etc. which are managed by the Component Builder. The 'screenshot' of the simulated environment is stored as Persim Project to allow the flexible and extendable design of the target space. The XML schema definition and a sample Persim project file is shown in Appendix C.

An Event-driven Simulation Engine runs the simulation and generates simulation events based on the configuration of the simulation project. The SDDL Generator is responsible for generating simulated data into the standardized SDDL [5], [6] format. Persim also provides SDDL Converter as an auxiliary tool to convert non SDDL data into SDDL format to analyzing other's dataset in a formal way. Lastly, Persim has a Project Renderer that captures the meta-data such as sensor information, space layout etc. from an SDDL file and creates the corresponding simulation project that represents the simulated environment of the actual space from where data was collected.

### Organization of the Simulation Model

Persim is developed as a component-based simulator in order to accommodate the various heterogeneous entities of the simulation environment. These components are used for defining (i) space to be simulated in terms of layout and sensors/actuators, (ii) activities to be simulated and (iii) simulation criteria/configuration. Each component is

characterized by several attributes. Now, we examine the major components of Persim in some details.

**Space**

This component captures the spatial aspects of the target simulated environment. It is currently in primitive stage, capturing only attributes such as space id and space name, giving no details to layouts, walkable and non-walkable areas, and to the precise relative locations among objects, sensors and actuators.

**Sensor**

Sensor component denotes the physical sensors that provide the contextual information such as movement, illumination, humidity etc. about the space. Motion sensor, light sensor, humidity sensor, Contact sensor etc are some examples of few commonly used sensors in home setting.

We have identified several attributes that are required to simulate the sensor in the virtual space of Persim. The major attributes are:

- **Id.** The identifier of the specific sensor instance.
- **Name.** The name of the specific sensor instance.
- **Type.** The type of the sensor is determined by the nature of its value generation. Persim allows three types of sensors for simulation:

  **Independent sensor.** A sensor which is triggered by any environment or nature. Temperature sensor is an example of an independent sensor since it generates temperature values regardless of any human activities around it. We also call the occurrence of independent sensor events as Time-driven event since it generated continuously over time. To simulate independent sensor, Persim requires the following

additional attributes related to the process generating function that generates the sensor event:

- **Process generation type.** Denotes whether the stepping function of the independent sensor event is based on fixed interval or probabilistic distribution.
- **Interval size.** The interval size in milliseconds if the process generation function is based on fixed interval.
- **Process generation function.** A function that defines the interval or stepping of sensor data generation. In other words, it defines how the sensor will be propelled with time. For example, the exponential distribution can be a process generation function.
- **Distribution parameters.** Denotes the parameters of the distribution such as mean, variance etc. depending on the type of the process generation function. For example, mean and variance are two parameters of the exponential distribution.

**Dependent sensor.** A sensor which is triggered by any external activity or human behavior. For example, pressure sensor is a dependent sensor since it only generates output when an external pressure is applied on the sensor. We have identified a special type of dependent sensor called 'Object Sensor', that requires separate explanation. Usually, when we refer to dependent sensor in Persim, we actually mean non-object dependent sensor.

**Object sensor.** A dependent sensor that triggers when any object attached to it is being handled or operated. For an example, an RFID tag on the TV remote control can be an object sensor. The object sensor is different from other dependent sensors because it has only two output values namely 'value when detected' and 'value when not detected' and its value generation process can be expressed as a Finite State Machine (FSM). Figure 4-2 shows a simple two-state FSM for the object sensor. In this figure, 'start' denotes the initial state and $p_{ii}$, $p_{ij}$, $p_{ji}$, and $p_{jj}$ are the transition probabilities that changes the state of the object sensor over time.

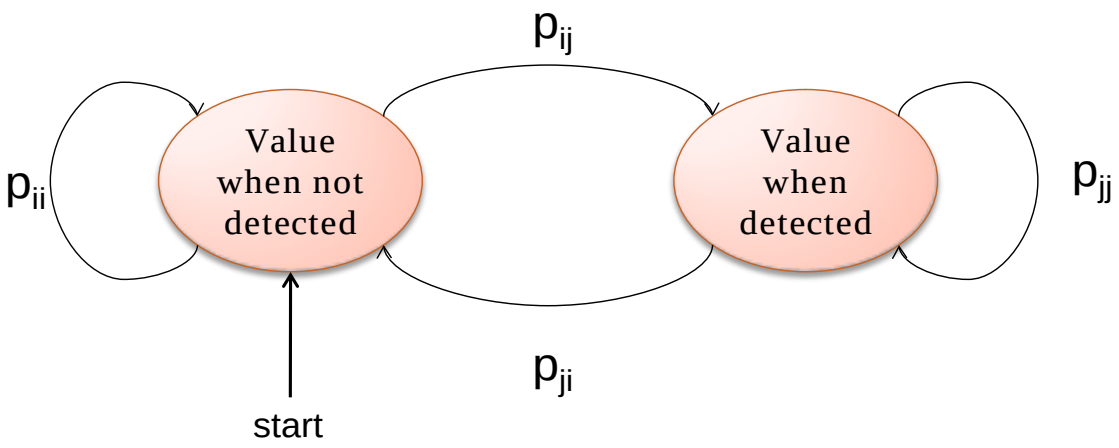


Figure 4-2. Finite state machine for object sensor

- **Location id.** Location identification for a specific sensor.
- **Functional type.** Type of the sensor based on the functionality. For example, motion and pressure denotes two different functional types of dependent sensors.
- **Minimum value.** The maximum value that a specific sensor can generate.
- **Maximum value.** The minimum value that a specific sensor can generate.
- **Initial value.** The initial value of a specific sensor before the experiment begins.
- **Value generation function.** A distribution function that characterizes the data generated by the sensor. For example, the value generation function for a motion sensor can be a binary valued function which generates value 1 when any movement of a person is detected and 0 otherwise.
- **Distribution parameters.** Denotes the parameters of the distribution such as mean, variance, etc. depending on the type of the value generation function. For example, upper bound and lower bound are two parameters of the uniform distribution function.

**Actuator**

Actuator component denotes the actuators or actors in the space that changes the state of the environment by acting on sensors or some objects. Servo, Electric motor, etc. are a few examples of commonly used actuators. A great majority of existing datasets use only sensors and no actuators. However datasets like the iDrom dataset of

University of Essex [18] use an actuator as another device deployed in the space. In this case, actuator component must be configured. Persim requires the following attributes for actuators, which are similar to those of sensors:

- **Id.** The identifier of the specific actuator instance.
- **Name.** The name of the specific actuator instance.
- **Location id.** Location identification for a specific actuator.
- **Functional type.** Type of the actuator based on the functionality. For example, servo and motor denotes two different functional types of actuators.
- **Minimum value.** The maximum value that a specific actuator can generate.
- **Maximum value.** The minimum value that a specific actuator can generate
- **Initial value.** The initial value of a specific actuator before the experiment starts.
- **Value generation function.** A distribution function that characterizes the data generated by the actuator. For example, the value generation function for a servo can be a uniform distribution which generates value from 0 to 360 degrees.
- **Distribution parameters.** Denotes the parameters of the distribution such as mean, variance, etc. depending on the type of the value generation function. For example, upper bound and lower bound are two parameters of uniform distribution function.

**Activity**

Activity is an independent Persim event that captures a human activity (e.g., walking to kitchen, getting up, leaving house, etc.) It is the basic event that acts as a stimulus and propels the simulation engine. Activities are generally sensed by several sensors and are characterized by following attributes:

- **Id.** The identifier of the specific activity instance.
- **Name.** The name of the specific activity instance.
- **Type.** The type/category of the specific activity such as house- keeping, food preparation, etc.

- **Interval type.** Denotes whether the stepping/inter-arrival function of the activity is based on fixed interval or probabilistic distribution.
- **Interval size.** The interval size in milliseconds if the inter-arrival function is based on fixed interval.
- **Inter-arrival function.** A function that defines the interval or stepping of a particular activity. It defines how the activity is generated with time. It is similar to the process generation function of the independent sensor.
- **Distribution parameter.** Denotes the parameters of the distribution such as mean, variance, etc. depending on the type of the inter-arrival function.

**Activity-Sensor Mapping**

An activity can be detected by one or more sensors. To define the semantics of the activity in terms of corresponding sensors, an Actuator-Sensor Mapping table is required. The mapping captures the dependency of a specific activity on a specific set of sensors by a causal relationship. Also, it is able to customize the ordering of sensor trigger and minimum/maximum value of a sensor for a particular activity. The details of this mapping are explained with an example in the Simulation Steps section. Figure 4-3 shows the concept of Activity-Sensor Mapping. The mapping function can be described mathematically as - $f_{Act\text{-}Map}$: $Act_i \rightarrow S$, where $Act_i$ is an activity and $S = \{S_1, S_2 ... S_k\}$ is the set of sensors that detect the activity.

**Actuator-Sensor Mapping**

This component is needed to be configured only when actual space has some actuators. Actuators act as a domain manipulator of the sensors. Typically, single sensor events such as values, ranges, etc., or the combination of multiple sensor events, trigger an actuator. It then shifts the environment from one state to another which again changes the value of some sensors.

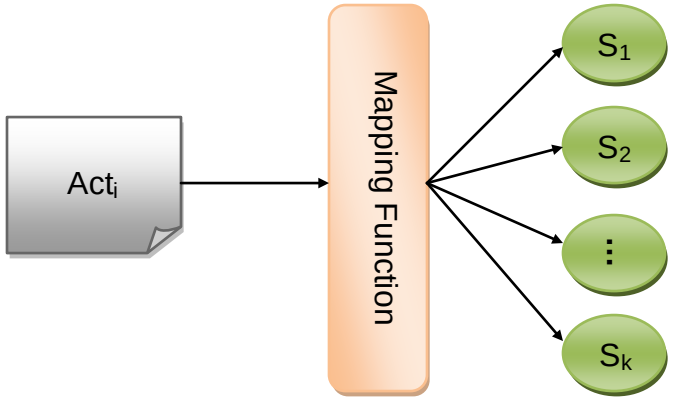


Figure 4-3. Activity-Sensor mapping

An Actuator-Sensor Mapping table describes the relationship between the actuator and sensors by two logistic functions: $M_1$ and $M_2$. The design of the Actuator-Sensor Mapping table is shown in Figure 4-4. In the figure, $M_1$ defines how a set of sensors triggers an actuator and $M_2$ how the triggered actuator changes the system state by affecting another set of sensors. Mathematically, the combined mapping function can be written as:

$$f_{Actu\text{-}Map}: S \times Actu_i \rightarrow S' \quad (4\text{-}1)$$

In Equation 4-1, *S* is the set of sensors that affect the actuator $Actu_i$ and *S'* is the set of sensors that will be affected by the action of the actuator $Actu_i$. In Figure 4-4, $S = \{S_{a1}, S_{a2}...,S_{aj}\}$, $Actu_i$ is the corresponding actuator and $S' = \{S_{b1}, S_{b2}...,S_{bk}\}$.

**Simulation Configuration**

This component of the simulator manages all the basic configuration parameters needed by the simulation engine. The parameters are listed with some details:

- **Simulation start time.** The starting time of the simulation.
- **Simulation end time.** The finishing time of the simulation.

- **Activities to be simulated.** Denotes the list of activities that is to be simulated. The user can add all possible activities while defining the Activity component mentioned earlier and choose a sub set of activities from them to be simulated.

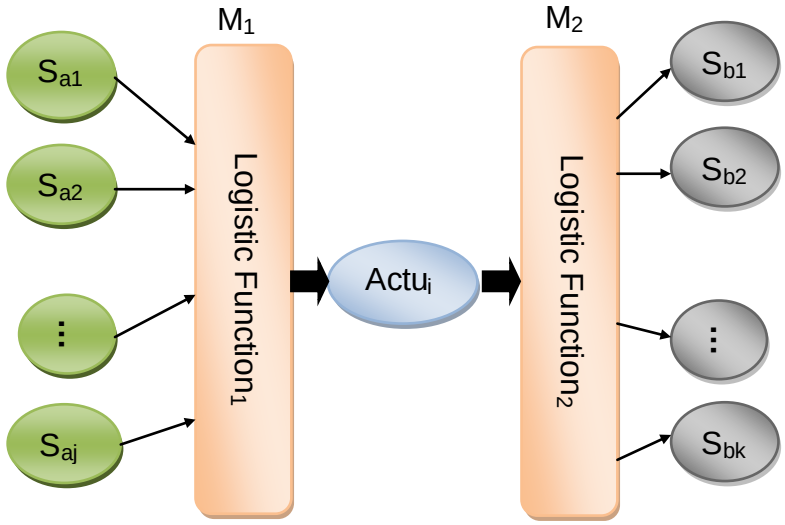


Figure 4-4. Actuator-Sensor mapping

- **Simulation mode.** Persim supports two simulation modes: Activity-driven and State space. In the Activity-driven mode, the user specifies the set of activities that happen in space. This mode activates the set of dependent sensors for all desired activities based on the Activity-Sensor Mapping. In State space mode, Persim generates a sequence of events reporting on all sensors (dependent and independent) deployed in the space irrespective of any activity. In this mode, the simulator records the state of the entire space and generated sensor events may or may not include any information about occurring activities.

- **Time-driven events to be simulated.** Denotes the list of events that are generated from independent sensors. The user can choose any number of independent sensors from the available sensor list and simulate according to his/her experimental requirement. Since these events are not triggered by any activity, they are simulated based on the process generation function of the corresponding sensor.

**Simulation Algorithm**

Persim adopts the discrete-event simulation model [24, pp. 1-103] using a next-event, time-advanced approach to capture the dynamic nature of the pervasive system which evolves over time. According to this classical simulation technique, the target state space changes whenever any event (either activity or time-driven) occurs and the

system variables of the space are updated based on the simulation logic. The flowchart of the simulation algorithm is shown in Figure 4-5. Persim simulation model consists of the following major variables:

- **Simulation clock (simClk).** A variable that keeps track of the current value of the logical simulated time. The domain of the clock value is the set of timestamps of all events. This simClk is always advanced only to the time of the next imminent event.
- **List of Events (eventList*(event, eventTime)*).** A list of tuples of the form (event, eventTime) where *event* denotes the simulation event time and eventTime denotes the time of the occurrence of that event. Each tuple represents an event trigger which needs to be processed by the simulator.
- **List of Activities (*activityList*).** A list of activities to be simulated selected by the user for a specific simulation project.
- **List of contexts (*contextList*).** A list of independent sensors selected by the user that act as "contexts" for the simulation.
- **Initialization Routine.** A method to initialize simClk and eventList before the simulation loop starts.
- **Timing Routine.** A method that determines the next event from the eventList and advances the simClk to the time of occurrence of the selected event.
- **Event Routine.** A method to process the current event based on its type. It also determines the time of occurrence of an event based on the library routine and re-schedules as needed.
- **Library Routine.** Consists of several methods that serve as utility functions for the simulation loop. These functions are used to generate random variables from the probability distribution of the inter-arrival time of an event and generation function of sensors (as specified by the user and configured by Persim).
- **Simulation Loop.** The main program that invokes the initialization routine, timing routine and event routine and that propels the simulation of the target space by processing and generating/scheduling new events for the entire lifecycle of the simulation.

  The simulation starts from the empty and idle state. First, it invokes the Initialization Routine to initialize the simClk and eventList. Then, the Timing Routine is invoked to find the most imminent (next) event from the eventList. It then advances the

simClk to the time of occurrence of the next event. It processes the event in the Event Routine according to the mode of simulation, whether it is Activity-driven or State space. The event routine also reschedules the current event to propel the simulation loop. The simulation continues until the simClk exceeds the designated simulation end time at which point the simulation loop exits.

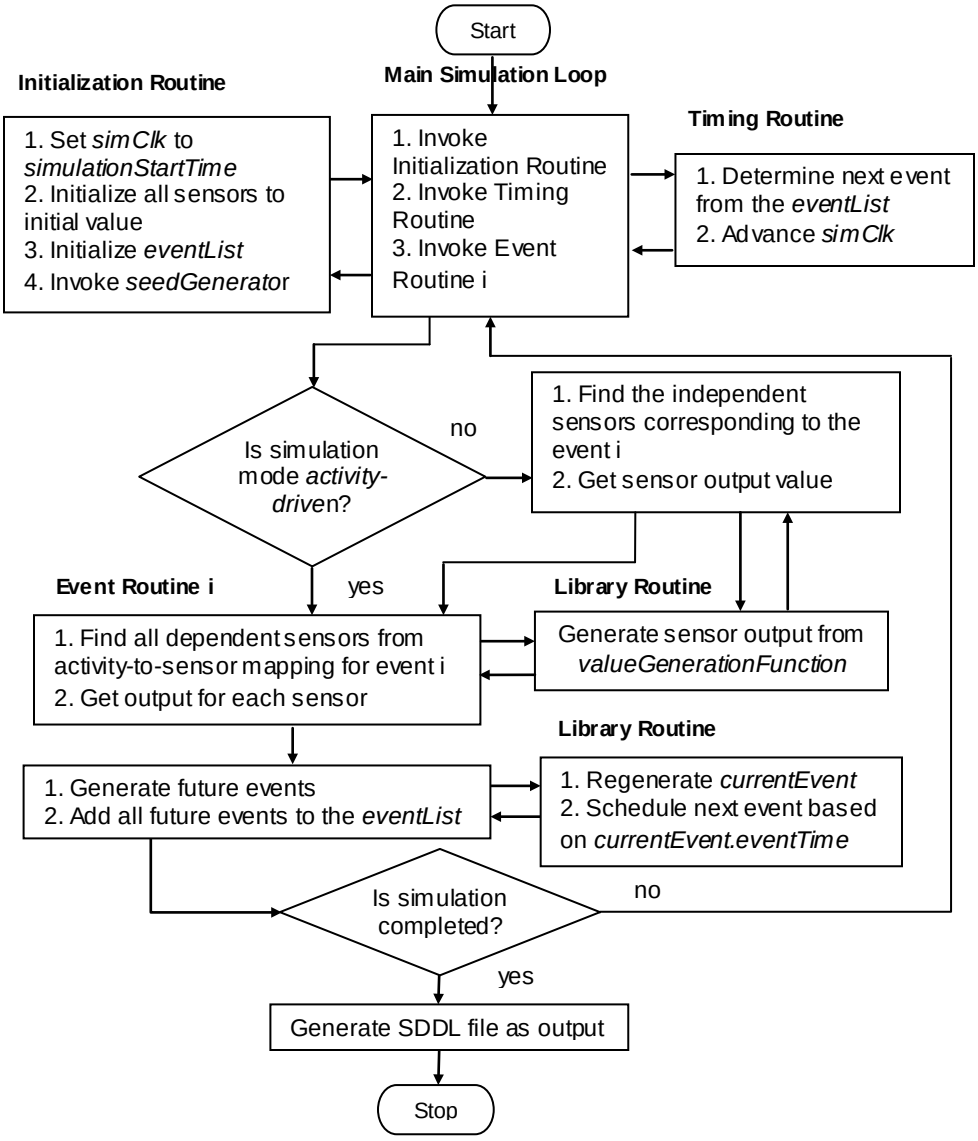


Figure 4-5. Flowchart of Persim event simulation

In Figure 4-6, the pseudo code shows the detail steps of the simulation. Here, lines 1-3 perform the initialization. Then the main simulation loop starts (lines 4-26) to

find the next imminent event, advance the simulation clock, process the event based on the simulation loop and re-generate the current event. The simulation loop continues as long as there are events in the eventList and the simulation clock does not exceed the simulation end time. The events are generated by the inter-arrival distribution function defined in the earlier steps. Finally, recorded simulated data is generated as an SDDL format.

```
Function simulationLoop
1. simClk ← simulationStateTime
2. eventList←initialize events from activityList and contextList
3. generate random seed for current simulation
4. while(simClk <= simulationEndTime)
5.  currentEvent ← find the most imminent event from eventList
6.  simClk ← currentEvent.eventTime
7.   if(simMode == activity_driven)
8.     processActivityDrivenEvent
9.       Determine dependent sensors from activityMapper
10.      Find next sensor to be triggered based on sensor seq#
11.      Record sensor output from valueGenerationFunction
12.  else     //simMode = state_space
13.     processStateSpaceEvent
14.       if(currentEvent.type == time_driven)
15.        Find independent sensors from contextList
16.        Record sensor output from valueGenerationFunction
17.       else    //currentEvent.type == activity
18.         processActivityDrivenEvent
19.           Perform the same steps as 9-11
20.       endif
21.  generateEvent
22.   Get newEventTime for currentEvent from inter-arrival
   distribution
23.   Schedule next event newEvent based on currentEvent.endTime
24.   Add the eligible newEvent to the eventList
25.  endif
26. endwhile
27. Generate SDDL file as output
```

Figure 4-6. Pseudo code of Persim simulation algorithm

### Simulation Steps

A researcher can design the simulation by configuring specifications of each of the components described earlier in the chapter. Firstly, the target physical space to be simulated (for example, smart home) has to be configured via the Space component.

The researcher can choose the layout (how many rooms) and name each area. Then the desired sensors and actuators can be deployed through the Sensor and Actuator components. Each sensor and actuator can be configured by specifying few parameters such as id, name, type, value generation function, min/max value, and so forth.

Activities are then added through the Activity component. Similar activity can have different impact based on the context of the activity. Thus, an activity can be re-named and configured with slight variations if necessary. For example, after adding activity 'walk', the researcher can change it to 'walk to the kitchen'.

Next, the researcher needs to map each activity to a set of dependent sensors using the Activity-Sensor Mapping table. In the mapping table, every sensor has a sequence number which represents the order of its trigger when an activity is being performed. The user can specify the sequence of sensor-triggers individually for each activity. For example, the motion sensor M1 for activity Act1has the sequence 1 and the motion sensor M2 has the sequence 2. This means M1 is always triggered before M2 while Act1 is being performed. Persim allows same sequence number for multiple sensors for the same activity since some sensors can be located together and sensed coincidently. Also, the sequence number can have 'don't care' value (represented as "#") which means the activity-detection does not depend on the ordering of that sensor-trigger [3].

Finally, the user must choose several simulation parameters such as simulation mode, simulation start and end time, activities to be simulated, etc. The two simulation modes, Activity-driven and State space, requires different configuration options. Although both simulations need information about the activities, only State space

requires setting up time-driven events. Note that time-driven events are defined by independent sensors configured in the Sensor component. If Activity-driven is selected, every single activity must be configured with attributes such as starting time, ending time, interval type, interval-arrival function, etc. For the interval type, Persim provides two options- constant interval or statistically distributed function. This completes all the information required for the simulation of events in the space and prepares the simulator to generate data [3]. Figure 4-7 shows the lifecycle of Persim simulation.

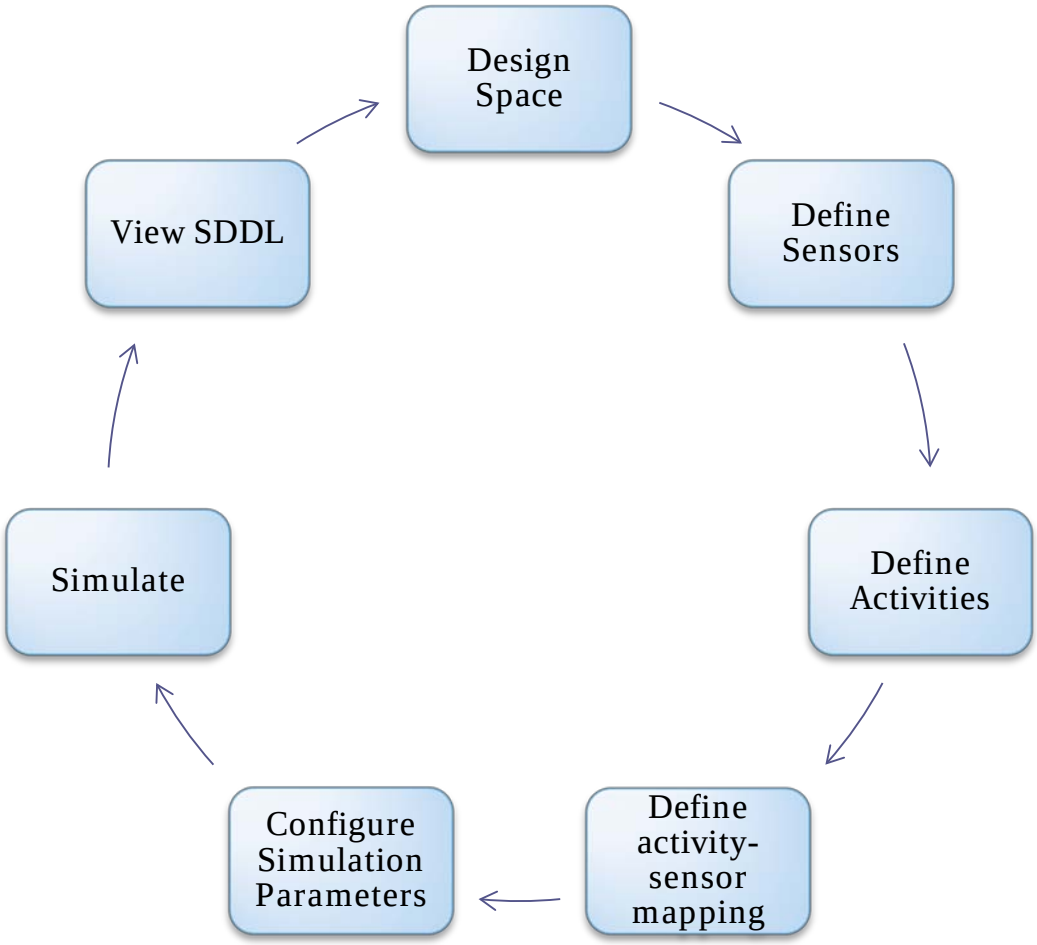


Figure 4-7. Persim simulation life cycle

## Simulation Output

The issue of representing the simulated data was also very important while developing the simulator. Unless there is a standard representation of the output data, our simulator cannot be fully utilized. Moreover, in order to support cross-platform use of Perism, the output has to be platform independent and flexible. Thus, the simulation output of Persim is generated as a XML file in SDDL format [5], [6] which is a proposed standard of dataset representation. By this, it can position the simulator as a tool that can 'speak' in a language understandable by the research communities. An example of the simulated SDDL file is shown in Appendix B.

## Development Environment

Persim is developed as a web-based simulator using Java, JavaScript and Java Server Pages (JSP) which are platform independent. The web application is hosted by the Apache Tomcat web server. Simulation project information and standardized output data are generated as XML files which are easy to parse and can be used cross-platform.

## Persim Usage Scenarios

### Create Focused Simulation

Persim can be used as a tool to create focused simulation in order to achieve particular research goal. This can save a tedious and time consuming process of running the experiment in the real space and collecting data for all possible test cases. For example, one can simulate two sets of data of 'wash hand' activity for a person with or without Alzheimer disease. In the former case, the dataset can have a water flow sensor running at the end of the activity since the patient may forget to turn on/off the water. In later case, the water flow sensor should be off almost all the time. These two

different sets of data can be very useful in the research of assisted living of patients with Alzheimer disease.

**Reproduce Experimental Data**

Under a particular experimental setup, same activity data can be reproduced by the Persim engine. A researcher has the flexibility to reuse the same simulation environment and generate data as many times as needed and at any extent. For example, a researcher may want to collect data for detecting the walking pattern of an obese person in the home setting. Often activity recognition algorithms do not perform well if sample data size is small. The researcher may find his/her test data to be too inadequate to detect the activity with reasonable accuracy. It may be infeasible and time consuming to recruit obese people for walking and re-do the experiment again to collect required data. In this situation, the researcher can use the collected data to create the simulation environment and reproduce the activity data with minimum effort.

**Modify Experimental Goal/Setup**

Persim allows the researcher to generate a simulated environment of the actual experimental setup by judiciously customizing essential component such as sensors, activities, and the semantics of the activity in terms of a set of sensors. It also allows one to design the simulated environment incrementally over multiple sessions. Thus, it empowers the owner of the dataset to go back in time and explore slight variations in the pervasive system without actually repeating the experiment and collecting additional data. Sometimes the physical space can have some issues that were found after collecting the data. For example, during the experiment one potential sensor may not work properly and produce erroneous data. Or a sensor can be missing in the experimental setup which is vital for particular activity recognition. In that case, Persim

can benefit the researcher by modifying the simulated environment as needed (by adding or deleting sensors) and generate data according to the specification.

**Extend Utility of the Dataset**

Data collected from a particular experimental setup is constrained by the hardware infrastructure and research-implicit goals/contexts. Depending on the event captured by the dataset, it may not be useful to some researchers without minor alterations. Persim is designed so that the user can load a SDDL file of an experiment and create the simulated environment from it. This can assist the researcher to modify the underlying experimental setup of the already collected data and explore additional goals which were not thought of during data collection. In this way, Persim can be useful to extend the utility of a dataset for further research. Thus Persim is intended to open a new dimension of collaborative research in the area of human activity recognition and ubiquitous computing applications.

**Create a Knowledge Repository of Datasets**

Persim has the capability of creating diverse simulation environment based on customizable parameters and generate human activities of interest. Also, by using Persim's powerful web-interface, one can produce a large volume of data within a very short time. Thus, Persim can be utilized to create specialized inventory of datasets similar to [8] and contribute to the critical problem of obtaining home-activity data. Therefore, instead of engaging huge of effort to build a smart home and collect the test data, researchers can focus on 'actual' research problem. By using the knowledge repository they can try different approaches quickly and cost-effectively and evaluate the accuracy of their early stage algorithms.

## Verification

To measure the level of fidelity between simulated pervasive space and its real counterpart in terms of their dataset, we have adopted a fuzzy-based verification technique [3], [15]. This was a collaborative effort with Dr. Hani Hagras from the University of Essex.

The data used to verify Persim was obtained from a smart apartment (whose layout is shown in Figure 4-8) in Washington State University (WSU) - part of the ongoing CASAS smart home project [17]. The collected data represents Activities of Daily Living (ADLs) which are considered as basic functionalities for independent living. Our verification process focused on the following activities:

- **Make a phone call (T1).** The participants are asked to look up a specified number in a phonebook, call the number, and write down the cooking directions spoken by the recorded message. The phonebook, notepad, and telephone are located on dining room table.
- **Wash hands (T2).** For this activity, participants are told to wash their hands in the kitchen sink using the soap and paper towels provided.
- **Clean (T5).** The cleaning activity required participants to clean the dishes and put the medicine bottle and other materials back into a cabinet.

We have created the simulated environment in Persim similar to that of CASAS smart home and generated 40-50 instances of simulated data for each of these three tasks. In order to verify the accuracy of Persim, the verification approach used the sensor data to construct a fuzzy logic based system (FLS) for both simulated and real dataset. It models the given process and generates the mapping as a set of rules from the data (real or simulated) to the correct activities/tasks. The motivation behind this approach is that only by using the data, we can generate fuzzy models which can be easily interpreted by the researcher [3], [15].

The verification process is accomplished in four phases and includes both black box and white box testing techniques [25, pp. 420-425]. First, Phase 1 performs the aggregation of the simulation and real data logs to generate the fuzzy sets for the simulated and real FLSs, respectively. In Phase 2, a sliding window approach is employed to generate the linguistic IF-Then rulebases for each FLSs. In Phase 3, a numerical verification is performed by feeding the real data to the simulation FLS to measure the similarity between the outputs of the simulated FLS and real data. This phase completes the black box testing process. In Phase 4, as a part of white box testing, the fuzzy sets and rule bases of both real and simulated FLSs are compared by linguistic verification. Figure 4-8 shows an overview of the fuzzy based verification for Persim.

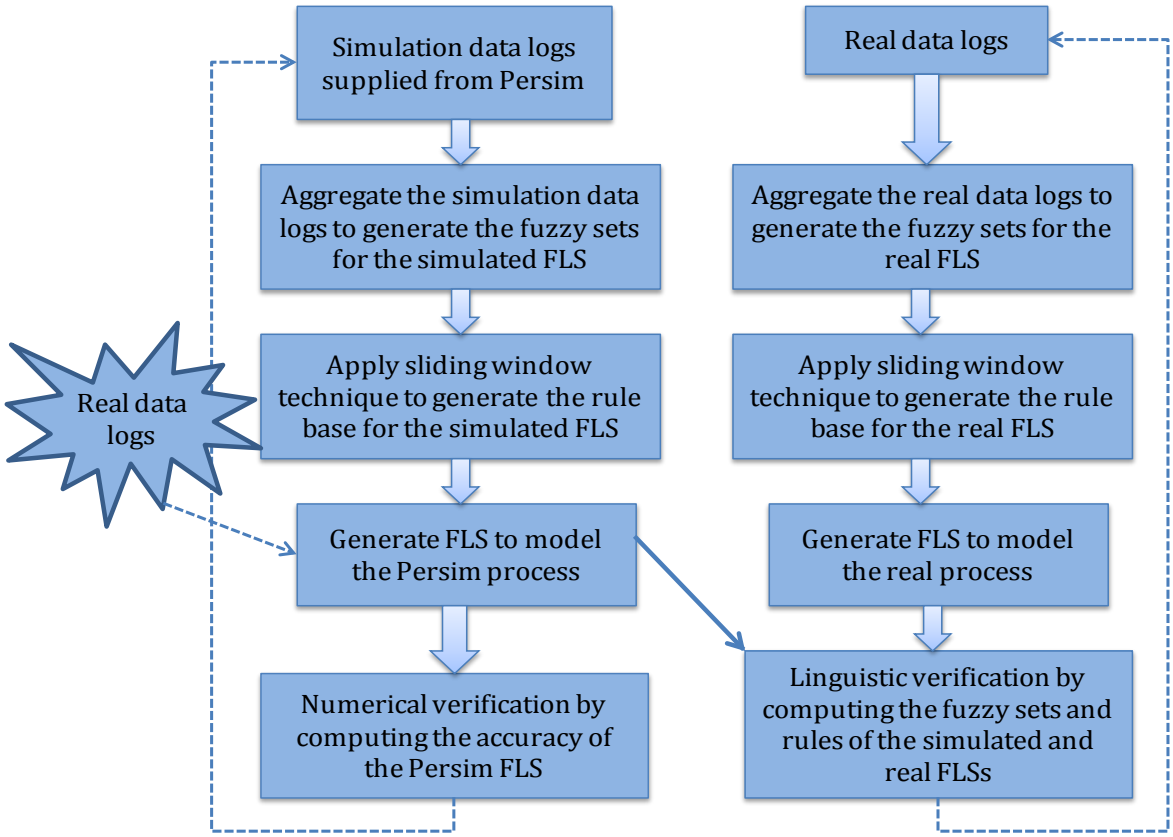


Figure 4-8. An overview of the fuzzy based verification approach [15]

**Results**

The percentage of accuracy of simulated data from Persim in predicting output activity obtained by validating against above mentioned test data is listed in Table 4-1. Here, the first column denotes the task/activity name; the second and third column represents the result obtained from phase 3 and phase 4 respectively.

Table 4-1. The verification results from WSU's CASAS smart home project [3], [15]

| Task name | Black box testing | White box testing |
|---|---|---|
| T1 (Make a phone call) | 70.83% | 99.8% |
| T2 (Wash hands) | 100% | 98.7% |
| T5 (Clean) | 41.67% | 99.8% |

**Analysis**

Based on the results shown in Table 4-1, we can claim that while comparing the rules of both real and simulated FLS (white box testing), all the simulated tasks performed well. However, when each real task instance was fed into the simulated FLS (black box testing), T5 task didn't perform well. We analyzed the activity to find the cause of this performance degradation.

More insights about the differences between the simulation and the real data can be discovered when investigating the rule bases of the FLS generated from the real world data and the FLS generated from the simulated data. We found that for this activity the user follows a path that has motion sensors M13, M14, M15, M16, and M17 placed in sequence. It was noted that the real FLS has shown this sequence. However, in simulation FLS has shown a sequence of M13 to M14 to M17, which is 'theoretically' not possible without triggering M15 and M16. Hence, the FLS generated from simulated data did not perform well, as in the case of T5.

Since in our simulation algorithm, for a particular activity, the sensor to be triggered next is selected randomly based on increasing ordering of the sequence

number, it is possible that M15 and M16 may not be selected which can result a sequence like M13->M14->M17. However, this situation can happen in practice where one sensor may fail to trigger when an activity is being performed. So we can relax the verification rules to capture this idea which will improve the accuracy percentage of the activity.

## Persim Features

### Allow Various Types of Sensors

Persim allows a wide variety sensors categorized into three groups-independent sensor, dependent non-object sensor and object sensor. Any sensor commonly used for human activity sensing can be placed in any of these three categories. Thus researcher can create the simulation environment of smart home with desired set of sensors.

### Support Different Modes of Simulation

Persim supports two simulation modes- Activity-driven and State-space. In Activity-driven simulation, one can zoom into the space and simulate only activities of interest similar to the CASAS smart apartment dataset of WSU [17]. Alternatively, one has the choice to simulate the entire state space regardless of any activities similar to the dataset of the iDorm project of University of Essex [18] in State-space mode. These simulation modes allows researcher to create different specialized datasets to achieve particular research objective.

### Sequential Activity Modeling

Persim can model activities that are performed in sequence. First, a complex activity is factorized into simple sequential sub–activities. For example, ‘Eat food’ activity can be divided into sub-activities such as ‘walk to the kitchen’, ‘take food from freezer’, ‘sit in the chair’ etc. Then it models each sub-activity by defining the causal

relationship between activity and the set of sensors that detects that activity in the target space. In the mapping, the user can also specify the order of sensor-trigger. Then user can choose the sequence of sub-activities that result the complex activity. Finally, user can select the appropriate inter-arrival function such as Poisson distribution, and Exponential distribution to define how activities will appear in the simulation environment in course of time.

**Allow Incremental Design**

In order to reuse the space design, all configuration information is stored in an XML-based project configuration file- known as Persim simulation project. There are several operations a user can perform during the life cycle of a simulation project such as saving current configuration, loading and deleting existing projects [3]. Also, a user can add, delete or modify any simulation components such as Sensor, Activity, and Simulation Configuration to alter simulation environment or goals and explore variations of the results. Based on simulation results, the user can go over the simulation steps again to make any modifications needed until desired results are reached. This empowers the researcher in refining the activity model and associated algorithms in an incremental fashion [2].

**Provide Toolbox for SDDL Conversion**

A text to SDDL conversion tool is provided with Persim to enable the conversion of existing datasets to SDDL format. By using its simple web-based interface, the owner of the dataset can provide meta-data about the experiment and sensor data and express it in SDDL format. It can be easily parsed by using publicly available SDDL parser and utilized as test data for various purposes. Since the standardized representation is key

to the methodological analysis of data, this feature can be a useful step in making human activity data re-usable and sharable across research groups.

# CHAPTER 5
# PERSIM CASE STUDY

In this chapter, we shall describe a human activity simulation scenario and show in step-by-step manner how we can use Persim to simulate the desired dataset.

## Problem Description

Let's assume our point of interest is to facilitate assisted living of individuals with Alzheimer disease. Often a person with Alzheimer disease forgets to complete essential tasks such as turn off the water tap and turn off the oven, which can cause potential danger. So, we would like to recognize such an activity so that we can take actions to ensure the safety of the resident. Suppose, we are focusing on detecting a simple cooking activity where the person walks from living room to kitchen, bakes the potato in the oven and washes hands. Now, the problem is how we can generate data that mimics these activities.

## Step-by-Step Simulation through Persim

In order to simulate the activities mentioned above, one has to go though some simple simulation steps those were described in the Chapter 4. First, we need to setup the space. For this particular scenario, we can create a 1X2 space with living room and kitchen. Now, we need to add sensors necessary sensors to capture the activities - 'walk to kitchen', 'bake potato', and 'wash hands'. Walking event is commonly detected by using motion sensors. So we can add motion sensors in the path of living room to kitchen. In order to detect 'bake potato' activity, we can attach contact sensor in the oven and motion sensor around the oven area of the kitchen. Finally, to detect 'wash hands', we can add water flow sensor in the kitchen sink and add motion sensors near the sink area.

After deploying sensors in the space, the simulation interface will look as shown in Figure 5-1. Here we have deployed 7 motion sensors M1 to M7, one object sensor ‘oven’ and one water flow sensor W1. While adding, each sensor can be configured with respective parameters such as value generation function, maximum value, and minimum value. For example, a reasonable value generation function for the motion sensor can be a binary distribution (i.e., when it generates 1 when any movement is detected and otherwise 0).

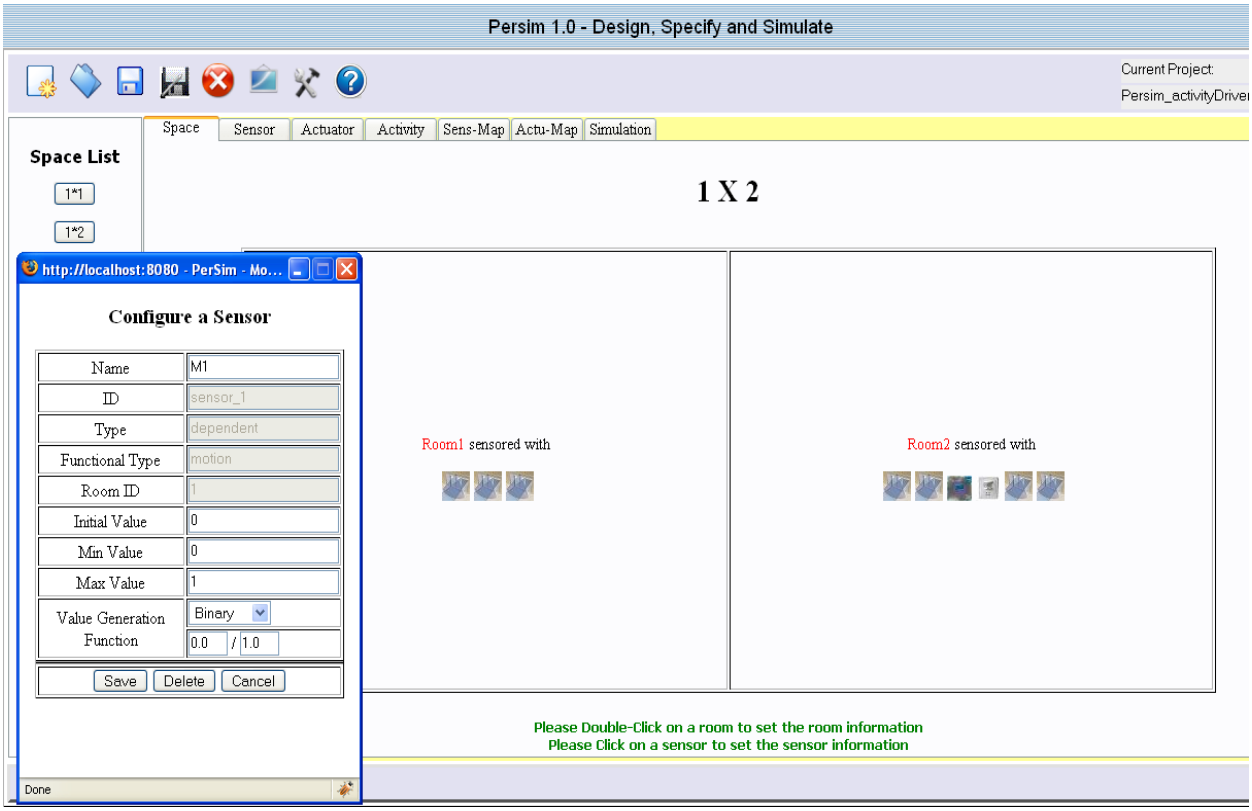


Figure 5-1. The screenshot of Persim simulation space while deploying sensors

Now we can add the desired activities that will be performed inside the space. According to our problem description activities are ‘walk to kitchen’ (named as walk), ‘bake potato’ (named as cook) and ‘wash hands’ (named as clean).

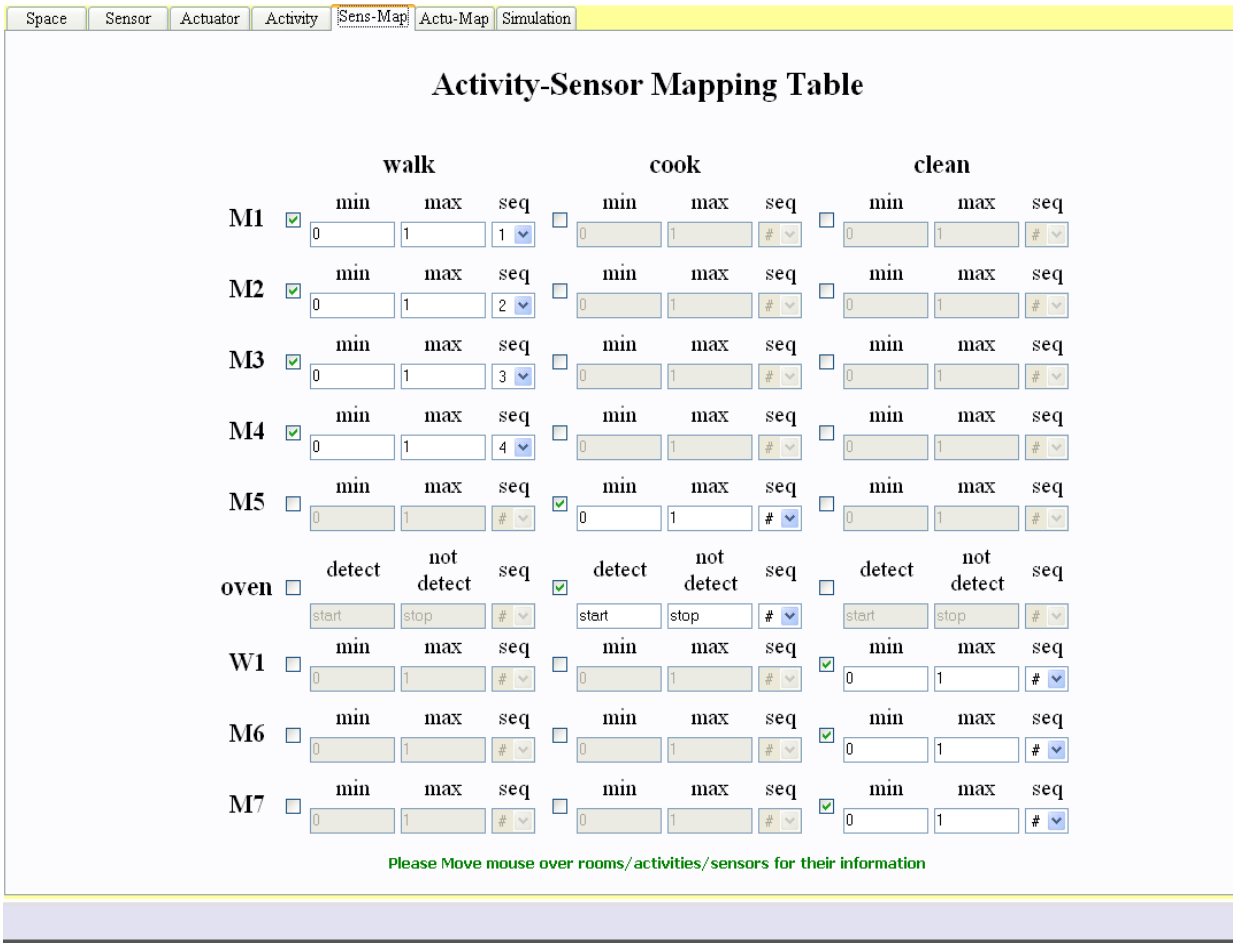


Figure 5-2. The activity-sensor mapping table

Next, we need to complete the mapping of activity to sensor as shown in Figure 5-2. Here the column denotes the activities to be simulated and the row denotes all available sensors in the space. While walking to the kitchen, suppose there are four motion sensors lined up in a sequence. Then we can select M1, M2, M3, and M4 in the walk column. Since they are lined up in a sequence, the order of sensor trigger will also be sequential. Thus we select 1, 2, 3, and 4 as the sequence number of M1, M2, M3, and M4 respectively. For the activity 'cook', suppose one object sensor 'oven' and one motion sensor 'M5' are involved. Since the handling of the oven and person's movement during cooking has no strict ordering, we can use '#' in the sequence number

field to express that sensor ordering is not important. Lastly, for ‘clean’ activity, suppose two motion sensors M6 and M7 and one water flow sensor, W1 are involved. Similar to previous activity, in this activity there is no strict ordering of sensor trigger.

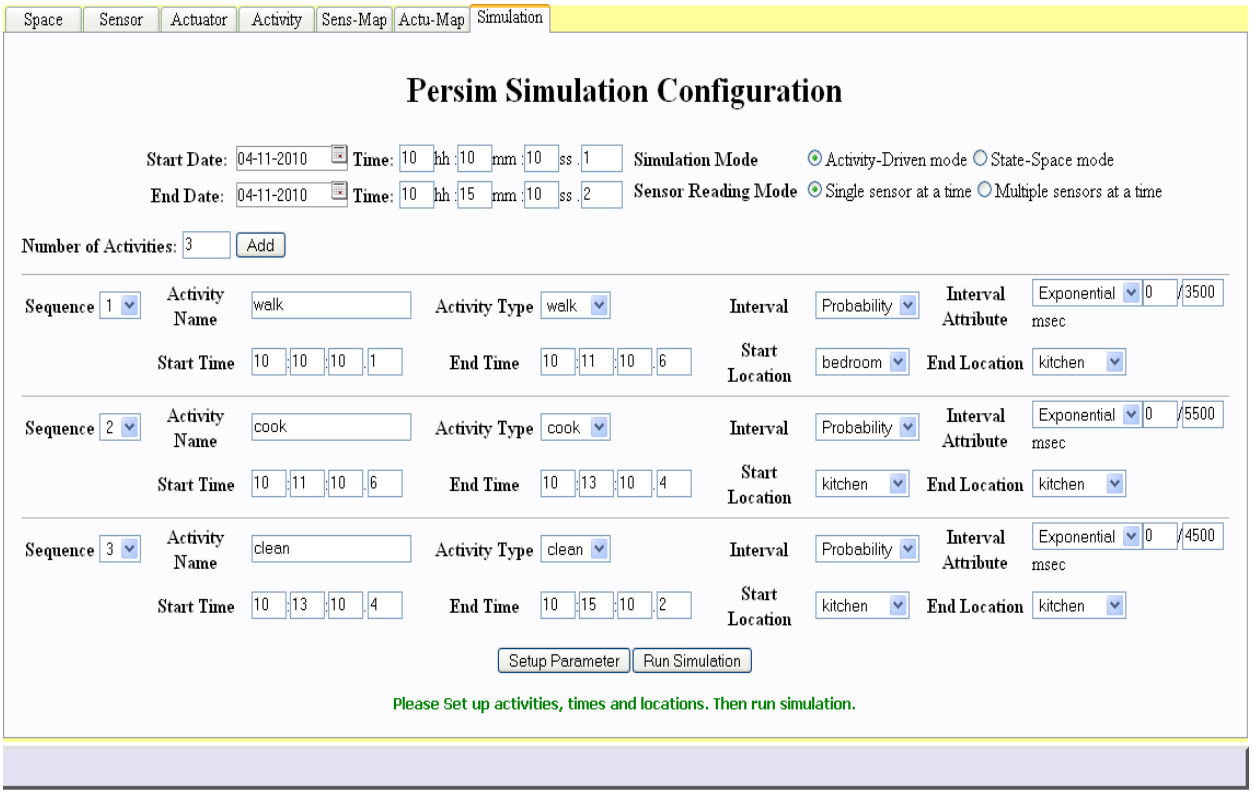


Figure 5-3. Persim simulation configuration interface

In the final step, we have to setup few simulation parameters. The corresponding Persim interface is showed in Figure 5-3. Here we can specify the start and end time of the simulation. Since, in the described scenario, the system is only affected by the activity of the person, the simulation mode should be set to ‘Activity-Driven mode’. We can choose sensor reading mode as either ‘single sensor at a time’ or ‘multiple sensor at a time’ based on the format of output we wish to have. Now, we can add activities to be simulated from previously defined activities and specify appropriate inter-arrival

distribution for each activity. Here we have chosen Exponential distribution with 0 mean and 3500 to 4500 milliseconds variance for three activities.

Now Persim is ready to generate activity data in SDDL format based on the above mentioned configuration. A sample of simulated SDDL data of the described scenario is shown in Appendix B.

# CHAPTER 6
# CONCLUSIONS AND FUTURE WORK

In this thesis, we presented Persim [2]-[4], as a tool to accelerate activity recognition research and save the tedious and time intensive effort required for deployment and integration of devices in an actual space. It is developed with a view to assist quick simulation of various human activities over multiple sessions and evaluate the performance of newly proposed ideas and algorithms in early stage. It can be utilized in determining the adequacy of the tentative sensor-set that was planned to be used to recognize a particular activity. Persim also empowers the researcher to go back in time and make virtual changes to the actual experimental setup and generate modified data that observes the changes. This can be done without repeating the experiment and thus avoids the huge time to collect data. As the simulation output, Persim promotes sharing of sensor data through an open standard - Sensory Dataset Description Language (SDDL) [5], [6], driven from a careful examination of several existing datasets. Persim has been well-verified both numerically and linguistically using a fuzzy based approach [3], [15], which compares the datasets simulated by Persim against the data collected from a real world deployment. We welcome any feedback or suggestions for improvements and especially additional features that the community values or desires. We also hope to engage the research community in refining the SDDL proposal, which is intended to be an open community standard.

Nevertheless, there are several features that can be added to improve Persim and utilize it in achieving greater research goals. Here, we suggest the following possibilities as future extensions:

- **Incorporate a labeling tool.** Researchers in activity recognition and context detection require annotated dataset to perform supervised and semi-supervised

learning. Persim can be extended to annotate both actual and simulated data to assist their research effort.

- **Include a verification tool.** Including the fuzzy-based verification tool in Persim can help the researcher assess the accuracy of their simulated data against similar real data. The verification tool can take one simulated dataset in SDDL and corresponding reference dataset in SDDL and act as a standalone tool for generating the percentage of accuracy. If the result is below some threshold, then the researcher can modify the simulation environment to generate data that is more close to the actual deployment.

- **Add space realism.** Currently, Persim has rudimentary space definitions with simple space layout and no relative locations among objects, sensors and actuator. By adding higher level of realism can place the simulator to capture more detail of the space and utilize the information to create more realistic simulation of activities. Space realism can be achieved through the use of space templates such as single family home, apartment, assisted living facility etc.

- **Support rules for actuator mapping.** Usually, an actuator is triggered by certain sensor events such as values, ranges or combination sensor events. An actuation, on the other hand, could affect the environment and cause a change in some sensor values (for example, turning light on may affect photo sensors). Realizing actuators in the space can allow Persim to create diverse scenarios where pervasive application logics such as 'activate alarm on potential danger', 'automatic lighting' are enabled. For this, we need to map actuators to sensors (described in Chapter 4) to define the causal-relationship among them. Persim can allow rule-based mapping that can help the researcher to easily simulate desired scenarios. The rules should be able to define any combination of actuator and sensors.

- **Apply FSM to model object sensor.** To model object sensor more accurately, Finite State Machine (FSM) (described in Chapter 4) can be added in Persim. The interface should allow the user to create an FSM for each object sensor with minimum effort. In simulation algorithm, Persim will maintain states for a particular object sensor and transition from one state to other based on the specified FSM.

- **Support automatic project generation from SDDL.** In designing Persim, one of our objectives was to import SDDL file from any real deployment and generate the simulation project automatically with minimum input from the user. This feature can be implemented so that researcher can easily analyze and generate data with slight variations to satisfy his/her research goals.

# APPENDIX A
# SDDL XML SCHEMA

```
<!-- Sensory Dataset Description Language(SDDL) Schema
        Version: 1.0
        Mobile and Pervasive Computing Lab, CISE Department, University of Florida
        Author: Shantonu Hossain (shossain@cise.ufl.edu)
        Project URL: http://www.icta.ufl.edu/projects_persim.htm
-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:sddl="http://www.example.org/SDDL_Schema"
targetNamespace="http://www.example.org/SDDL_Schema" elementFormDefault="qualified">
        <!-- Root element of SDDL schema version 1.0 -->
        <xsd:element name="Sensory_Dataset">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:element name="Contact_Info" minOccurs="0"/>
                                <xsd:element name="History" minOccurs="0"/>
                                <xsd:element name="Dataset_Contexts"/>
                                <xsd:element name="Sensor_Event"/>
                        </xsd:sequence>
                        <xsd:attribute name="version" type="xsd:string"/>
                        <xsd:attribute name="id" type="xsd:string"/>
                        <xsd:attribute name="name" type="xsd:string"/>
                        <xsd:attribute name="date_from" type="xsd:string"/>
                        <xsd:attribute name="date_to" type="xsd:string"/>
                        <!-- Major elements of Sensory_Dataset -->
                        <!-- attributes of Sensory_Dataset such as version, name, date etc. -->
                </xsd:complexType>
        </xsd:element>
        <!-- Element: Contact_Info -->
        <!-- Contains information of the authorized person of the dataset.-->
        <xsd:complexType name="Contact_Info">
                <xsd:attribute name="name" type="xsd:string"/>
                <xsd:attribute name="role" type="xsd:string"/>
                <xsd:attribute name="organization" type="xsd:string"/>
                <xsd:attribute name="phone" type="xsd:int"/>
                <xsd:attribute name="email" type="xsd:string"/>
        </xsd:complexType>
        <!-- Element: History -->
        <!-- Contains meta-data and relevant information about the project and the dataset.-->
        <xsd:complexType name="History">
                <xsd:sequence>
                        <!-- A brief description of the project -->
                        <xsd:element name="Project_Description" type="xsd:string" minOccurs="0"/>
                        <!-- A brief description of the dataset -->
                        <xsd:element name="Dataset_Description" type="xsd:string" minOccurs="0"/>
                        <!-- A brief description of the space from where the sensor data is collected. -->
                        <xsd:element name="Space_Layout" type="xsd:string" minOccurs="0"/>
                        <xsd:element name="Project_Specification"/>
                </xsd:sequence>
        </xsd:complexType>
        <!-- Element: Project_Specification -->
        <!-- Contains meta-data and relevant information about the project and the dataset.-->
        <xsd:complexType name="Project_Specification">
```

```
        <xsd:sequence>
                <xsd:element name="Location_Info" minOccurs="0"/>
                <xsd:element name="Sensor_Info"/>
                <xsd:element name="Actuator_Info" minOccurs="0"/>
                <xsd:element name="Activity_Info" minOccurs="0"/>
                <xsd:element name="Active_Contexts" minOccurs="0"/>
                <xsd:element name="Subject_Info" minOccurs="0"/>
        </xsd:sequence>
</xsd:complexType>
<!-- Contains meta-data about the locations in the space.-->
<xsd:complexType name="Location_Info">
        <xsd:sequence>
                <xsd:element name="Location" minOccurs="0" maxOccurs="unbounded">
                        <xsd:complexType>
                                <xsd:attribute name="id" type="xsd:string"/>
                                <xsd:attribute name="name" type="xsd:string"/>
                                <xsd:attribute name="zone" type="xsd:string"/>
                        </xsd:complexType>
                </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="count" type="xsd:int" default="0"/>
</xsd:complexType>
<!-- Contains meta-data about sensors that are deployed in the space.-->
<xsd:complexType name="Sensor_Info">
        <xsd:sequence>
                <xsd:element name="Sensor" maxOccurs="unbounded">
                <xsd:complexType>
                <xsd:attribute name="id" type="xsd:string" use="required"/>
                <xsd:attribute name="name" type="xsd:string"/>
                <xsd:attribute name="type" type="xsd:string"/>
                <xsd:attribute name="location_id" type="xsd:string"/>
                <xsd:attribute name="unit" type="xsd:string"/>
                <xsd:attribute name="min_value" type="xsd:double" use="optional" default="0"/>
                <xsd:attribute name="max_value" type="xsd:double" use="optional" default="1"/>
                </xsd:complexType>
                </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="count" type="xsd:int" use="required"/>
</xsd:complexType>
<!-- Contains meta-data about actuators that are deployed in the space.-->
<xsd:complexType name="Actuator_Info">
        <xsd:sequence>
                <xsd:element name="Actuator" minOccurs="0" maxOccurs="unbounded">
                        <xsd:complexType>
                                <xsd:attribute name="id" type="xsd:string"/>
                                <xsd:attribute name="name" type="xsd:string"/>
                                <xsd:attribute name="type" type="xsd:string"/>
                                <xsd:attribute name="location_id" type="xsd:string"/>
                                <xsd:attribute name="unit" type="xsd:string"/>
                                <xsd:attribute name="min_value" type="xsd:double"/>
                                <xsd:attribute name="max_value" type="xsd:double"/>
                        </xsd:complexType>
                </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="count" type="xsd:int" default="0"/>
</xsd:complexType>
```

```
	<!-- Contains meta-data about activities that are performed during the experiment.-->
	<xsd:complexType name="Activity_Info">
		<xsd:sequence>
			<xsd:element name="Activity" minOccurs="0" maxOccurs="unbounded">
				<xsd:complexType>
					<xsd:attribute name="id" type="xsd:string"/>
					<xsd:attribute name="name" type="xsd:string"/>
					<xsd:attribute name="category" type="xsd:string"/>
					<xsd:attribute name="sub-category" type="xsd:string"/>
				</xsd:complexType>
			</xsd:element>
		</xsd:sequence>
		<xsd:attribute name="count" type="xsd:int" default="0"/>
	</xsd:complexType>
	<!-- Contains meta-data about active contexts related to the dataset such as 'the age of the
subject', 'month of the year' etc.-->
	<xsd:complexType name="Active_Contexts">
		<xsd:sequence>
			<xsd:element name="Active_Context" minOccurs="0" maxOccurs="unbounded">
				<xsd:complexType>
					<xsd:attribute name="id" type="xsd:string"/>
					<xsd:attribute name="name" type="xsd:string"/>
					<xsd:attribute name="type" type="xsd:string"/>
				</xsd:complexType>
			</xsd:element>
		</xsd:sequence>
		<xsd:attribute name="count" type="xsd:int" default="0"/>
	</xsd:complexType>
	<!-- Contains meta-data about the human-subject who participated in the experiment.-->
	<xsd:complexType name="Subject_Info">
		<xsd:sequence>
			<xsd:element name="Subject" minOccurs="0" maxOccurs="unbounded">
			<xsd:complexType>
			<xsd:attribute name="id" type="xsd:string"/>
			<xsd:attribute name="name" type="xsd:string"/>
			<xsd:attribute name="age" type="xsd:double"/>
			<xsd:attribute name="ethnicity" type="xsd:string"/>
			<xsd:attribute name="gender" type="xsd:string"/>
			<xsd:attribute name="disease_or_disability_condition1" type="xsd:string"/>
			<xsd:attribute name="disease_or_disability_condition2" type="xsd:string"/>
			<xsd:attribute name="disease_or_disability_condition3" type="xsd:string"/>
				</xsd:complexType>
			</xsd:element>
		</xsd:sequence>
		<xsd:attribute name="count" type="xsd:int" default="0"/>
	</xsd:complexType>
	<!-- Element: Dataset_Contexts -->
	<!-- Contains information about the organization and representation of the dataset such as
parameters, separator of the parameters etc.-->
	<xsd:complexType name="Dataset_Contexts">
		<xsd:sequence>
<!-- Contains the parameters and its index presented in the dataset e.g. parameters can be timestamp,
sensorID, sensorValue etc. -->
			<xsd:element name="Parameters">
			<xsd:complexType>
				<xsd:sequence>
```

```
                                        <xsd:element name="Parameter" maxOccurs="unbounded">
                                                <xsd:complexType>
                                                        <xsd:attribute name="name" type="xsd:string"/>
                                                        <xsd:attribute name="index" type="xsd:int"/>
                                                        </xsd:complexType>
                                                </xsd:element>
                                        </xsd:sequence>
                                        <xsd:attribute name="count" type="xsd:int" default="0"/>
                                </xsd:complexType>
                        </xsd:element>
<!-- Contains the different separators for parameters, groups and lines e.g. comma, semicolon, space etc.
-->
                        <xsd:element name="Separators">
                        <xsd:complexType>
                                <!-- Separators between parameters in the dataset -->
                                <xsd:attribute name="parameter_separator" type="xsd:string"/>
                                <!-- Separators between two lines/data instances in the dataset -->
                                <xsd:attribute name="line_separator" type="xsd:string"/>
                </xsd:complexType>
                                </xsd:element>
                                <!-- Contains the type of the sensor event -->
                                <xsd:element name="Sensor_Event_Type">
                                <xsd:simpleType>
                                        <xsd:restriction base="xsd:string">
                                                <xsd:enumeration value="Multiple_Sensors_per_Event"/>
                                                <xsd:enumeration value="Single_Sensor_per_Event"/>
                                        </xsd:restriction>
                                </xsd:simpleType>
                                </xsd:element>
                        </xsd:sequence>
        </xsd:complexType>
        <!-- Element: Sensor_Event -->
        <!-- Contains actual sensor data collected from the space.-->
        <xsd:complexType name="Sensor_Event">
                <xsd:sequence>
                        <xsd:element name="Event" maxOccurs="unbounded">
                        <xsd:complexType>
                        <xsd:attribute name="data" type="xsd:string" use="required"/>
                        <xsd:attribute name="activity_performed" type="xsd:string" use="optional"/>
                        <xsd:attribute name="active_context" type="xsd:string" use="optional"/>
                        </xsd:complexType>
                        </xsd:element>
                </xsd:sequence>
        </xsd:complexType>
</xsd:schema>
```

APPENDIX B
SAMPLE SDDL

**Example of a SDDL file of WSU Smart Apartment's Dataset**

This is an example SDDL file generated by SDDL Converter for '*t03.t2*' data file collected from WSU Apartment Testbed [17] for normal ADL activities.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Sensory_Dataset xmlns="http://www.example.org/SDDL_Schema" version="1" name="p03.t2" id="" date_to="02-29-2008" date_from="02-29-2008">
   <Contact_Info xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Contact_Info" phone="" organization="Washington State University" name="Dr. Diane J. Cook" email=""/>
   <History xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="History">
      <Project_Description>The experiment is performed in WSU Smart Apartment is part of the ongoing CASAS smart home project at WSU. The CASAS project treats environments as intelligent agents, where the status of the residents and their physical surroundings are perceived using sensors and the environment is acted upon using controllers in a way that improves the comfort, safety, and/or productivity of the residents</Project_Description>
      <Dataset_Description>These datasets represent sensor events collected in the WSU smart apartment testbed. The data represents participants performing five ADL activities in the apartment.</Dataset_Description>
      <Space_Layout>The apartment is a three-bedroom apartment located on the Washington State University campus. It includes three bedrooms, one bathroom, a kitchen, and a living / dining room.  The apartment is equipped with motion sensors distributed approximately 1 meter apart throughout the space.</Space_Layout>
      <Project_Specification xsi:type="Project_Specification">
         <Location_Info xsi:type="Location_Info" count="0"/>
         <Sensor_Info xsi:type="Sensor_Info" count="7">
            <Sensor unit="" type="water sensor" name="sensor7" min_value="0.0" max_value="1.0" location_id="" id="AD1-B"/>
            <Sensor unit="" type="motion sensor" name="sensor6" min_value="0.0" max_value="1.0" location_id="" id="M18"/>
            <Sensor unit="" type="motion sensor" name="sensor5" min_value="0.0" max_value="1.0" location_id="" id="M17"/>
            <Sensor unit="" type="motion sensor" name="sensor4" min_value="0.0" max_value="1.0" location_id="" id="M16"/>
            <Sensor unit="" type="motion sensor" name="sensor3" min_value="0.0" max_value="1.0" location_id="" id="M15"/>
            <Sensor unit="" type="motion sensor" name="sensor2" min_value="0.0" max_value="1.0" location_id="" id="M14"/>
            <Sensor unit="" type="motion sensor" name="sensor1" min_value="0.0" max_value="1.0" location_id="" id="M13"/>
         </Sensor_Info>
         <Actuator_Info xsi:type="Actuator_Info" count="0"/>
         <Activity_Info xsi:type="Activity_Info" count="5">
            <Activity sub-category="" name="clean" id="act_5" category=""/>
            <Activity sub-category="" name="eat" id="act_4" category=""/>
            <Activity sub-category="" name="cook" id="act_3" category=""/>
            <Activity sub-category="" name="wash hand" id="act_2" category=""/>
            <Activity sub-category="" name="make phone call" id="act_1" category=""/>
         </Activity_Info>
         <Active_Contexts xsi:type="Active_Contexts" count="0"/>
```

```
        <Subject_Info xsi:type="Subject_Info" count="0"/>
      </Project_Specification>
   </History>
   <Dataset_Contexts xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Dataset_Contexts">
      <Parameters count="3">
        <Parameter name="Sensor_Value" index="3"/>
        <Parameter name="Sensor_Id/Name" index="2"/>
        <Parameter name="Timestamp" index="1"/>
      </Parameters>
      <Separators parameter_separator="space" line_separator="new line"/>
      <Sensor_Event_Type>Single_Sensor_per_Event</Sensor_Event_Type>
   </Dataset_Contexts>
   <Sensor_Event xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Sensor_Event">
      <Event data="2008-02-29 13:23:49.819614 M14 ON
      2008-02-29 13:23:50.838368 M15 ON
      2008-02-29 13:23:51.432171 M16 ON
      2008-02-29 13:23:52.592843 M17 ON
      2008-02-29 13:23:52.87184 M14 OFF
      2008-02-29 13:23:53.9793 M13 OFF
      2008-02-29 13:23:54.169416 M16 OFF
      2008-02-29 13:23:54.541335 M15 OFF
      2008-02-29 13:23:55.16146 AD1-B 0.0369928
      2008-02-29 13:23:56.74917 M17 OFF
      2008-02-29 13:24:13.7882 AD1-B 0.173948
      2008-02-29 13:24:15.532252 M17 ON
      2008-02-29 13:24:18.814592 M17 OFF
      2008-02-29 13:24:19.427606 M17 ON
      2008-02-29 13:24:19.525 AD1-B 0.0929702
      2008-02-29 13:24:24.105685 M17 OFF
      2008-02-29 13:24:24.247656 M17 ON
      2008-02-29 13:24:28.424866 M18 ON
      2008-02-29 13:24:33.443316 M18 OFF
      2008-02-29 13:24:34.829277 M17 OFF
      2008-02-29 13:24:35.609198& M17 ON" activity_performed="wash hand" active_context=""/>
   </Sensor_Event>
</Sensory_Dataset>
```

**Example of a SDDL file of Persim Simulation**

This SDDL file is generated by Persim which corresponds to the scenario described in Chapter 5.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Sensory_Dataset xmlns="http://www.example.org/SDDL_Schema" name="activtyDriven" id="2">
   <History xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="History">
      <Project_Description>Sample activityDriven project for PerSim</Project_Description>
      <Dataset_Description>This dataset corresponds to 3 activities. A person walks to the kitchen, bakes
potato and washes hands.</Dataset_Description>
      <Project_Specification xsi:type="Project_Specification">
        <Sensor_Info xsi:type="Sensor_Info" count="9">
          <Sensor type="dependent" min_value="0.0" max_value="1.0" id="M1"/>
          <Sensor type="dependent" min_value="0.0" max_value="1.0" id="M2"/>
          <Sensor type="dependent" min_value="0.0" max_value="1.0" id="M3"/>
```

```
            <Sensor type="dependent" min_value="0.0" max_value="1.0" id="M4"/>
            <Sensor type="dependent" min_value="0.0" max_value="1.0" id="M5"/>
            <Sensor type="dependent" min_value="0.0" max_value="1.0" id="oven"/>
            <Sensor type="dependent" min_value="0.0" max_value="1.0" id="W1"/>
            <Sensor type="dependent" min_value="0.0" max_value="1.0" id="M6"/>
            <Sensor type="dependent" min_value="0.0" max_value="1.0" id="M7"/>
         </Sensor_Info>
         <Activity_Info xsi:type="Activity_Info" count="3">
            <Activity name="walk" id="act_1"/>
            <Activity name="cook" id="act_2"/>
            <Activity name="clean" id="act_3"/>
         </Activity_Info>
      </Project_Specification>
   </History>
   <Dataset_Contexts xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Dataset_Contexts">
      <Parameters count="3">
         <Parameter name="Timestamp" index="1"/>
         <Parameter name="Sensor_Id" index="2"/>
         <Parameter name="Sensor_Value" index="3"/>
      </Parameters>
      <Separators parameter_separator="space" line_separator="new line"/>
      <Sensor_Event_Type>Single_Sensor_per_Event</Sensor_Event_Type>
   </Dataset_Contexts>
   <Sensor_Event xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Sensor_Event">
      <Event data="2010-04-28 10:10:10.844 M1 1.0
      2010-04-28 10:10:22.189 M2 1.0
      2010-04-28 10:10:23.167 M2 0.0
      2010-04-28 10:10:25.301 M2 1.0
      2010-04-28 10:10:26.822 M3 1.0
      2010-04-28 10:10:28.5 M3 0.0
      2010-04-28 10:10:29.544 M4 1.0" activity_performed="walk"/>
      <Event data="2010-04-28 10:11:13.539 M5 1.0
      2010-04-28 10:11:29.556 M5 0.0
      2010-04-28 10:11:33.206 M5 1.0
      2010-04-28 10:11:39.95 M5 0.0
      2010-04-28 10:11:41.681 M5 1.0
      2010-04-28 10:11:50.149 M5 0.0
      2010-04-28 10:11:51.963 M5 1.0
      2010-04-28 10:12:02.967 oven 0
      2010-04-28 10:12:02.967 oven stop
      2010-04-28 10:12:06.692 M5 0.0
      2010-04-28 10:12:15.639 oven 1
      2010-04-28 10:12:15.639 oven start
      2010-04-28 10:12:15.8 M5 1.0
      2010-04-28 10:12:17.821 M5 0.0
      2010-04-28 10:12:28.902 M5 1.0
      2010-04-28 10:12:39.896 oven 0
      2010-04-28 10:12:39.896 oven stop
      2010-04-28 10:12:43.478 M5 0.0
      2010-04-28 10:12:49.108 oven 1
      2010-04-28 10:12:49.108 oven start
      2010-04-28 10:12:54.833 M5 1.0
      2010-04-28 10:12:58.505 oven 0
      2010-04-28 10:12:58.505 oven stop&
      2010-04-28 10:13:10.038 M5 0.0" activity_performed="cook"/>
```

```
        <Event data="2010-04-28 10:13:10.599 M7 1.0
        2010-04-28 10:13:10.662 M6 1.0
        2010-04-28 10:13:11.071 M7 0.0
        2010-04-28 10:13:17.364 M6 0.0
        2010-04-28 10:13:18.139 M6 1.0
        2010-04-28 10:13:28.026 M6 0.0
        2010-04-28 10:13:28.507 M6 1.0
        2010-04-28 10:13:30.024 M7 1.0
        2010-04-28 10:13:33.983 M6 0.0
        2010-04-28 10:13:38.141 W1 0.52942
        2010-04-28 10:13:39.622 W1 0.20786
        2010-04-28 10:13:50.166 M6 1.0
        2010-04-28 10:13:58.825 M6 0.0
        2010-04-28 10:14:02.957 M6 1.0
        2010-04-28 10:14:03.494 M6 0.0
        2010-04-28 10:14:05.536 M6 1.0
        2010-04-28 10:14:10.869 M6 0.0
        2010-04-28 10:14:11.153 M6 1.0
        2010-04-28 10:14:24.431 W1 0.0919
        2010-04-28 10:14:34.126 M6 0.0
        2010-04-28 10:14:36.361 M7 0.0
        2010-04-28 10:14:36.414 M6 1.0
        2010-04-28 10:14:49.005 W1 0.0603
        2010-04-28 10:14:58.162 M6 0.0
        2010-04-28 10:15:08.73 M6 1.0" activity_performed="clean"/>
    </Sensor_Event>
</Sensory_Dataset>
```

APPENDIX C
PERSIM PROJECT FILE

## Persim Project Schema

```
<!-- Persim Project Schema
        Version: 1.0
        Mobile and Pervasive Computing Lab, CISE Department, University of Florida
        Author: Shantonu Hossain (shossain@cise.ufl.edu)
              Jae Woong Lee (jwlee@cise.ufl.edu)
        Project URL: http://www.icta.ufl.edu/projects_persim.htm
-->
<xsd:schema xmlns:xsd ="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/Components"
xmlns:tns="http://www.example.org/Components" elementFormDefault="qualified">
<!-- Root element of Persim project schema version 1.0 -->
<xsd:element name="Simulation_Project">
        <xsd:complexType>
                <xsd:sequence>
                <!-- Major elements of Simulation_Project -->
                        <xsd:element name="Project_Info" minOccurs ="1"
maxOccurs="1"></xsd:element>
                        <xsd:element name="Sensor_Info" minOccurs ="1"
maxOccurs="1"></xsd:element>
                        <xsd:element name="Actuator_Info" minOccurs ="1"
maxOccurs="1"></xsd:element>
                        <xsd:element name="Activity_Info" minOccurs ="1"
maxOccurs="1"></xsd:element>
                        <xsd:element name="Space_Info" minOccurs ="0"
maxOccurs="1"></xsd:element>
                        <xsd:element name="Mapping" minOccurs ="0" maxOccurs="1"></xsd:element>
                        <xsd:element name="Simulation_Config" minOccurs ="0"
maxOccurs="1"></xsd:element>
                        <xsd:element name="Parameter_Config" minOccurs ="0"
maxOccurs="1"></xsd:element>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>

<!-- Element: Project_Info -->
<!-- Contains information about the simulation project i.e. name, id, version etc.-->
<xsd:complexType name="Project_Info">
        <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
        <xsd:attribute name="version" type="xsd:string"></xsd:attribute>
        <xsd:attribute name="date_created" type="xsd:string"></xsd:attribute>
        <xsd:attribute name="date_last_modified" type="xsd:string"></xsd:attribute>
        <xsd:attribute name="author" type="xsd:string"></xsd:attribute>
        <xsd:attribute name="path" type="xsd:string"></xsd:attribute>
</xsd:complexType>

<!-- Element: Sensor_Info -->
<!-- Contains information about the deployed sensors in the simulation space.-->
<xsd:complexType name="Sensor_Info">
        <xsd:sequence>
                <xsd:element name="Sensor" minOccurs="0" maxOccurs="unbounded">
```

```
                <xsd:complexType>
                        <xsd:attribute name="id" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="room_id" type="xsd:int"></xsd:attribute>
                        <xsd:attribute name="type" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="category" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="object" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="value_detected" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="value_not_detected" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="init_val" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="min_val" type="xsd:double"></xsd:attribute>
                        <xsd:attribute name="max_val" type="xsd:double"></xsd:attribute>
                        <xsd:attribute name="sensing_range" type="xsd:int"></xsd:attribute>

                        <xsd:attribute name="generation_function" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="mean" type="xsd:double"></xsd:attribute>
                        <xsd:attribute name="variance" type="xsd:double"></xsd:attribute>
                        <xsd:attribute name="sens_proc_type" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="sens_proc_interval" type="xsd:double"></xsd:attribute>
                        <xsd:attribute name="sens_proc_distr" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="sens_proc_mean" type="xsd:double"></xsd:attribute>
                        <xsd:attribute name="sens_proc_var" type="xsd:double"></xsd:attribute>

                </xsd:complexType>
                </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="count" type="xsd:int" default="0"></xsd:attribute>
</xsd:complexType>

<!-- Element: Actuator_Info -->
<!-- Contains information about the deployed actuators in the simulation space.-->
<xsd:complexType name="Actuator_Info">
        <xsd:sequence>
                <xsd:element name="Actuator" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                        <xsd:attribute name="id" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="room_id" type="xsd:int"></xsd:attribute>
                        <xsd:attribute name="type" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="init_val" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="generation_function" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="mean" type="xsd:double"></xsd:attribute>
                        <xsd:attribute name="variance" type="xsd:double"></xsd:attribute>
                </xsd:complexType>
                </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="count" type="xsd:int" default="0"></xsd:attribute>
</xsd:complexType>

<!-- Element: Activity_Info -->
<!-- Contains information about the activities in the simulation space.-->
<xsd:complexType name="Activity_Info">
        <xsd:sequence>
                <xsd:element name="Activity" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                        <xsd:attribute name="id" type="xsd:string"></xsd:attribute>
```

```
                        <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="type" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="start_time" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="end_time" type="xsd:string"></xsd:attribute>
                </xsd:complexType>
                </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="count" type="xsd:int" default="0"></xsd:attribute>
</xsd:complexType>

<!-- Element: Space_Info -->
<!-- Contains information about the layout of the simulation space.-->
<xsd:complexType name="Space_Info">
<xsd:sequence>
        <xsd:element name="Space" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
                <xsd:attribute name="id" type="xsd:string"></xsd:attribute>
                <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
                <xsd:attribute name="area" type="xsd:string"></xsd:attribute>
        </xsd:complexType>
        </xsd:element>
</xsd:sequence>
<xsd:attribute name="count" type="xsd:int" default="0"></xsd:attribute>
<xsd:attribute name="row" type="xsd:int" default="0"></xsd:attribute>
<xsd:attribute name="col" type="xsd:int" default="0"></xsd:attribute>
</xsd:complexType>

<!-- Element: Mapping -->
<!-- Contains the activity to sensor mapping and actuator to sensor mapping information.-->
<xsd:complexType name="Mapping">
<xsd:sequence>
        <xsd:element name="Map_Activity" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
                <xsd:sequence>
                                <xsd:element name="Map_Sensor" minOccurs="0"
maxOccurs="unbounded">
                                <xsd:complexType>
                                        <xsd:attribute name="sen_id" type="xsd:string"></xsd:attribute>
                                        <xsd:attribute name="sen_max"
type="xsd:string"></xsd:attribute>
                                        <xsd:attribute name="sen_min"
type="xsd:string"></xsd:attribute>
                                        <xsd:attribute name="sen_seq" type="xsd:int"></xsd:attribute>
                                </xsd:complexType>
                                </xsd:element>
                                <xsd:element name="Map_Space" minOccurs="0"
maxOccurs="unbounded">
                                <xsd:complexType>
                                                <xsd:attribute name="space_id"
type="xsd:string"></xsd:attribute>
                                </xsd:complexType>
                                </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="act_id" type="xsd:string"></xsd:attribute>
        </xsd:complexType>
        </xsd:element>
```

```
        <xsd:element name="Map_Actuator" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="Map_Sensor" minOccurs="0"
maxOccurs="unbounded">
                        <xsd:complexType>
                                <xsd:attribute name="sen_id" type="xsd:string"></xsd:attribute>
                                <xsd:attribute name="value_from"
type="xsd:string"></xsd:attribute>
                                <xsd:attribute name="value_to"
type="xsd:string"></xsd:attribute>
                                <xsd:attribute name="operator"
type="xsd:string"></xsd:attribute>
                        </xsd:complexType>
                        </xsd:element>
                        <xsd:element name="Map_Space" minOccurs="0"
maxOccurs="unbounded">
                        <xsd:complexType>
                                        <xsd:attribute name="space_id"
type="xsd:string"></xsd:attribute>
                        </xsd:complexType>
                        </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="actu_id" type="xsd:string"></xsd:attribute>
        </xsd:complexType>
        </xsd:element>
</xsd:sequence>
</xsd:complexType>

<!-- Element: Simulation_Config -->
<!-- Contains information about the simulation configuration of the dataset such as simulation start time,
activities to be simulated etc.-->
<xsd:complexType name="Simulation_Config">
<xsd:sequence>
        <xsd:element name="Sim_Activity" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="act_intervalType">
                        <xsd:complexType>
                                <xsd:choice>
                                        <xsd:element name="constant">
                                        <xsd:complexType>
                                                <xsd:attribute name="constant_value"
type="xsd:int"></xsd:attribute>
                                        </xsd:complexType>
                                        </xsd:element>
                                        <xsd:element name="probability">
                                        <xsd:complexType>
                                                <xsd:attribute name="probability_type"
type="xsd:string"></xsd:attribute>
                                                <xsd:attribute name="probability_mean"
type="xsd:int"></xsd:attribute>
                                                <xsd:attribute name="probability_variance"
type="xsd:int"></xsd:attribute>
                                        </xsd:complexType>
```

```
                                        </xsd:element>
                                    </xsd:choice>
                                    <xsd:attribute name="act_interval_type"
type="xsd:string"></xsd:attribute>
                                </xsd:complexType>
                                </xsd:element>
                        </xsd:sequence>
                        <xsd:attribute name="act_seq" type="xsd:int"></xsd:attribute>
                        <xsd:attribute name="act_name" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="act_type" type="xsd:string"></xsd:attribute>

                        <xsd:attribute name="act_start_hour" type="xsd:int"></xsd:attribute>
                        <xsd:attribute name="act_start_min" type="xsd:int"></xsd:attribute>
                        <xsd:attribute name="act_start_sec" type="xsd:int"></xsd:attribute>
                        <xsd:attribute name="act_start_mil" type="xsd:double"></xsd:attribute>
                        <xsd:attribute name="act_end_hour" type="xsd:int"></xsd:attribute>
                        <xsd:attribute name="act_end_min" type="xsd:int"></xsd:attribute>
                        <xsd:attribute name="act_end_sec" type="xsd:int"></xsd:attribute>
                        <xsd:attribute name="act_end_mil" type="xsd:double"></xsd:attribute>
                        <xsd:attribute name="act_start_location" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="act_end_location" type="xsd:string"></xsd:attribute>
                </xsd:complexType>
                </xsd:element>
                <xsd:element name="Sim_Indendent_Event" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                        <xsd:attribute name="event_name" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="event_type" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="event_index" type="xsd:int"></xsd:attribute>
                </xsd:complexType>
                </xsd:element>
</xsd:sequence>
<xsd:attribute name="start_date" type="xsd:string"></xsd:attribute>
<xsd:attribute name="end_date" type="xsd:string"></xsd:attribute>
<xsd:attribute name="start_hour" type="xsd:int"></xsd:attribute>
<xsd:attribute name="start_min" type="xsd:int"></xsd:attribute>
<xsd:attribute name="start_sec" type="xsd:int"></xsd:attribute>
<xsd:attribute name="start_mil" type="xsd:int"></xsd:attribute>
<xsd:attribute name="end_hour" type="xsd:int"></xsd:attribute>
<xsd:attribute name="end_min" type="xsd:int"></xsd:attribute>
<xsd:attribute name="end_sec" type="xsd:int"></xsd:attribute>
<xsd:attribute name="end_mil" type="xsd:int"></xsd:attribute>
<xsd:attribute name="num_activities" type="xsd:int"></xsd:attribute>
<xsd:attribute name="num_independent_events" type="xsd:int"></xsd:attribute>
<xsd:attribute name="sensor_read_mode" type="xsd:string"></xsd:attribute>
<xsd:attribute name="simultation_mode" type="xsd:string"></xsd:attribute>
</xsd:complexType>

<!-- Element: Parameter_Config -->
<!-- Contains information about the parameters in the simulated dataset.-->
<xsd:complexType name="Parameter_Config">
<xsd:sequence>
        <xsd:element name="dataset_parameter" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                        <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
                        <xsd:attribute name="index" type="xsd:int"></xsd:attribute>
                </xsd:complexType>
```

```
        </xsd:element>
</xsd:sequence>
<xsd:attribute name="dataset_name" type="xsd:string"></xsd:attribute>
<xsd:attribute name="dataset_ID" type="xsd:string"></xsd:attribute>
<xsd:attribute name="project_description" type="xsd:string"></xsd:attribute>
<xsd:attribute name="dataset_description" type="xsd:string"></xsd:attribute>
<xsd:attribute name="dataset_separator" type="xsd:string"></xsd:attribute>
<xsd:attribute name="dataset_line_separator" type="xsd:string"></xsd:attribute>
<xsd:attribute name="dataset_param_num" type="xsd:int"></xsd:attribute>
</xsd:complexType>
</xsd:schema>
```

## Sample Persim Project File

A sample Persim project file that corresponds to the simulation environment described in Chapter 5.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Simulation_Project xmlns="http://www.example.org/Components">
   <Project_Info xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Project_Info"
name="Persim_activityDriven"/>
   <Sensor_Info xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Sensor_Info"
count="9">
      <Sensor variance="0.0" value_not_detected="" value_detected="" type="dependent"
sens_proc_var="0.0" sens_proc_type="" sens_proc_mean="0.0" sens_proc_interval="0.0"
sens_proc_distr="" room_id="1" object="" name="M1" min_val="0.0" mean="0.0" max_val="1.0"
init_val="0" id="sensor1" generation_function="binary" category="motion"/>
      <Sensor variance="0.0" value_not_detected="" value_detected="" type="dependent"
sens_proc_var="0.0" sens_proc_type="" sens_proc_mean="0.0" sens_proc_interval="0.0"
sens_proc_distr="" room_id="1" object="" name="M2" min_val="0.0" mean="0.0" max_val="1.0"
init_val="0" id="sensor2" generation_function="binary" category="motion"/>
      <Sensor variance="0.0" value_not_detected="" value_detected="" type="dependent"
sens_proc_var="0.0" sens_proc_type="" sens_proc_mean="0.0" sens_proc_interval="0.0"
sens_proc_distr="" room_id="1" object="" name="M3" min_val="0.0" mean="0.0" max_val="1.0"
init_val="0" id="sensor3" generation_function="binary" category="motion"/>
      <Sensor variance="0.0" value_not_detected="" value_detected="" type="dependent"
sens_proc_var="0.0" sens_proc_type="" sens_proc_mean="0.0" sens_proc_interval="0.0"
sens_proc_distr="" room_id="2" object="" name="M4" min_val="0.0" mean="0.0" max_val="1.0"
init_val="0" id="sensor4" generation_function="binary" category="motion"/>
      <Sensor variance="0.0" value_not_detected="" value_detected="" type="dependent"
sens_proc_var="0.0" sens_proc_type="" sens_proc_mean="0.0" sens_proc_interval="0.0"
sens_proc_distr="" room_id="2" object="" name="M5" min_val="0.0" mean="0.0" max_val="1.0"
init_val="0" id="sensor5" generation_function="binary" category="motion"/>
      <Sensor variance="0.0" value_not_detected="stop" value_detected="start" type="dependent"
sens_proc_var="0.0" sens_proc_type="" sens_proc_mean="0.0" sens_proc_interval="0.0"
sens_proc_distr="" room_id="2" object="oven" name="oven" min_val="0.0" mean="0.0" max_val="1.0"
init_val="stop" id="sensor6" generation_function="" category="object"/>
      <Sensor variance="1.0" value_not_detected="" value_detected="" type="dependent"
sens_proc_var="0.0" sens_proc_type="" sens_proc_mean="0.0" sens_proc_interval="0.0"
sens_proc_distr="" room_id="2" object="" name="W1" min_val="0.0" mean="0.0" max_val="1.0"
init_val="0" id="sensor7" generation_function="uniform" category="light"/>
      <Sensor variance="0.0" value_not_detected="" value_detected="" type="dependent"
sens_proc_var="0.0" sens_proc_type="" sens_proc_mean="0.0" sens_proc_interval="0.0"
sens_proc_distr="" room_id="2" object="" name="M6" min_val="0.0" mean="0.0" max_val="1.0"
```

```
init_val="0" id="sensor8" generation_function="binary" category="motion"/>
      <Sensor variance="0.0" value_not_detected="" value_detected="" type="dependent"
sens_proc_var="0.0" sens_proc_type="" sens_proc_mean="0.0" sens_proc_interval="0.0"
sens_proc_distr="" room_id="2" object="" name="M7" min_val="0.0" mean="0.0" max_val="1.0"
init_val="0" id="sensor9" generation_function="binary" category="motion"/>
   </Sensor_Info>
   <Actuator_Info xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Actuator_Info"
count="0"/>
   <Activity_Info xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Activity_Info"
count="3">
      <Activity name="walk" id="act_1"/>
      <Activity name="cook" id="act_2"/>
      <Activity name="clean" id="act_3"/>
   </Activity_Info>
   <Space_Info xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Space_Info" row="1"
count="2" col="2">
      <Space name="bedroom" id="room1"/>
      <Space name="kitchen" id="room2"/>
   </Space_Info>
   <Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Mapping">
      <Map_Activity act_id="act_1">
         <Map_Sensor sen_seq="1" sen_min="0" sen_max="1" sen_id="sensor1"/>
         <Map_Sensor sen_seq="2" sen_min="0" sen_max="1" sen_id="sensor2"/>
         <Map_Sensor sen_seq="3" sen_min="0" sen_max="1" sen_id="sensor3"/>
         <Map_Sensor sen_seq="4" sen_min="0" sen_max="1" sen_id="sensor4"/>
      </Map_Activity>
      <Map_Activity act_id="act_2">
         <Map_Sensor sen_seq="0" sen_min="0" sen_max="1" sen_id="sensor5"/>
         <Map_Sensor sen_seq="0" sen_min="start" sen_max="stop" sen_id="sensor6"/>
      </Map_Activity>
      <Map_Activity act_id="act_3">
         <Map_Sensor sen_seq="0" sen_min="0" sen_max="1" sen_id="sensor7"/>
         <Map_Sensor sen_seq="0" sen_min="0" sen_max="1" sen_id="sensor8"/>
         <Map_Sensor sen_seq="0" sen_min="0" sen_max="1" sen_id="sensor9"/>
      </Map_Activity>
   </Mapping>
   <Simulation_Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Simulation_Config" start_sec="10" start_min="10" start_mil="1" start_hour="10" start_date="04-
11-2010" simultation_mode="Activity_Driven" sensor_read_mode="Single_Sensor_per_Event"
num_independent_events="0" num_activities="3" end_sec="10" end_min="15" end_mil="2"
end_hour="10" end_date="04-11-2010">
      <Sim_Activity act_type="walk" act_start_sec="10" act_start_min="10" act_start_mil="1.0"
act_start_location="bedroom" act_start_hour="10" act_seq="1" act_name="walk" act_end_sec="10"
act_end_min="11" act_end_mil="6.0" act_end_location="kitchen" act_end_hour="10">
         <act_intervalType act_interval_type="probDist">
            <probability probability_variance="3500" probability_type="exponential" probability_mean="0"/>
         </act_intervalType>
      </Sim_Activity>
      <Sim_Activity act_type="cook" act_start_sec="10" act_start_min="11" act_start_mil="6.0"
act_start_location="kitchen" act_start_hour="10" act_seq="2" act_name="cook" act_end_sec="10"
act_end_min="13" act_end_mil="4.0" act_end_location="kitchen" act_end_hour="10">
         <act_intervalType act_interval_type="probDist">
            <probability probability_variance="5500" probability_type="exponential" probability_mean="0"/>
         </act_intervalType>
      </Sim_Activity>
      <Sim_Activity act_type="clean" act_start_sec="10" act_start_min="13" act_start_mil="4.0"
```

```
act_start_location="kitchen" act_start_hour="10" act_seq="3" act_name="clean" act_end_sec="10"
act_end_min="15" act_end_mil="2.0" act_end_location="kitchen" act_end_hour="10">
          <act_intervalType act_interval_type="probDist">
             <probability probability_variance="4500" probability_type="exponential" probability_mean="0"/>
          </act_intervalType>
       </Sim_Activity>
    </Simulation_Config>
    <Parameter_Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Parameter_Config" project_description="Sample activityDriven project for PerSim"
dataset_separator="space" dataset_param_num="3" dataset_name="sample1"
dataset_line_separator="new line" dataset_description="sample description" dataset_ID="2">
       <dataset_parameter name="Timestamp" index="1"/>
       <dataset_parameter name="Sensor_Id" index="2"/>
       <dataset_parameter name="Sensor_Value" index="3"/>
    </Parameter_Config>
</Simulation_Project>
```

LIST OF REFERENCES


[1] T. Abdelzaher, Cyber Physical Computing Site [Internet]. Department of Computer Science, University of Illinois at Urbana Champaign; [updated 2010 May 18; cited 2010 May 18] Available from: http://www.cs.uiuc.edu/homes/zaher/cyberphysical/intro.html.

[2] S. Hossain, A. Helal, JW Lee and H. Hagras, "PerSim: Pervasive Space Simulation," in *Proc. Pervasive 2010 Poster Program*, Helsinki, Finland, May 17-20, 2010.

[3] S. Hossain, A. Helal, J.W. Lee, H. Hagras, A. Alfaham and D. Cook, "Persim – A Simulator for Human Activities in Pervasive Spaces," in Proc. *12th ACM Int. Conf. Ubiquitous Computing*, Copenhagen, Denmark, Sep. 26-29, 2010 (under review).

[4] Persim Project Website [Internet]. Computer Science and Information Science and Engineering Department, University of Florida; [updated 2009 Nov 19; cited 2010 May 18]. Available from: http://www.icta.ufl.edu/projects_persim.htm.

[5] Helal, A. Mendez-Vazquez, S. Hossain, "Specification and Synthesis of Sensory Datasets in Pervasive Spaces," in *Proc. IEEE Symposium of Computers and Communications*, pp. 694-698, Tunis, Tunisia, Jul. 2009.

[6] Sensory Dataset Description Language (SDDL) Specification and Website [Internet]. Computer Science and Information Science and Engineering Department, University of Florida; [updated 2009 Nov 19; cited 2010 May 18]. Available from: http://www.icta.ufl.edu/persim/sddl/.

[7] Open Geospatial Consortium, Inc. Site [Internet]. Sensor Model Language (SensorML); [updated 2010 May 18; cited 2010 May 18]. Available from: http://vast.uah.edu/SensorML.

[8] BoxLab Wiki Page [Internet]. Shared Inventory of Home Activity Dataset; [updated 2010 May 11; cited 2010 May 18]. Available from: http://boxlab.wikispaces.com/Mission.

[9] J. N. Al-Karaki and G. A. Al-Mashaqbeh, "SENSORIA, A New Simulation Platform for Wireless Sensor Networks," in *Proc IEEE Int. Conf. Sensor Technologies and Applications*, pp. 424-429, Oct. 2007.

[10] M. Varshney and R. Bagrodia, "Detailed Models for Sensor Network Simulations and their Impact on Network Performance," in *Proc. 7th ACM Int. Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems,* pp. 70-77, Oct. 2004.

[11] T. Antoine-Santoni, J. F. Santucci, E. De Gentili and B. Costa, “Modelling & Simulation oriented components of Wireless Sensor Network using DEVS Formalism,” in *Proc. Spring Simulation Multiconference,* vol. 2, pp. 299-306, Mar. 2007.

[12] M. C. Huebscher and J. A. McCann, “Simulation Model for Self-Adaptive Applications in Pervasive Computing,” in IEEE *Proc. Database and Expert Systems Applications, 15th International Workshop*, pp. 694-698, Aug. 2004.

[13] S. Nath, P. B. Gibbons, S. Seshan and Z. Anderson, “TOSSIM: accurate and scalable simulation of entire tinyos applications,” *ACM Trans. Sensor Networks (TOSN),* vol. 4, Mar. 2008.

[14] W. Jouve, J. Bruneau and C. Consel. DiaSim, “A Parameterized Simulator for Pervasive Computing Applications,” in *Proc. IEEE Int. Conf. Pervasive Computing and Communications,* vol. 00, pp. 1-3, Mar. 2009.

[15] Amr Elfaham, H. Hagras, A. Helal, S. Hossain, J. W. Lee, D. J. Cook, “A Fuzzy based Verification Agent for Persim Human Activity Simulator in Ambient Intelligent Environments,” in *Proc. IEEE World Congress on Computational Intelligence*, Barcelona, Spain, Jul. 18-23, 2010.

[16] I. Armac and D. Retkowitz, "Simulation of Smart Environments," in *Proc. IEEE Int. Conf. Pervasive Services*, pp. 257-266, Jul. 15-20, 2007.

[17] D. J. Cook and M. Schmitter-Edgecombe, “Activity Profiling using Pervasive Sensing in Smart Homes,” *IEEE Trans. Information Technology for Biomedicine*, 2008.

[18] V. Callaghan, J. Woods, S. Fitz, T. Dennis, H. Hagras, M. Colley, I. Henning, “The Essex iDorm: A Testbed for Exploring Intelligent Energy Usage Technologies in the Home,” in *Proc. Int. Conf. Intelligent Green and Energy Efficient Building and Technologies*, Beijing, China, Apr. 2008.

[19] E. Munguia Tapia, “Activity Recognition in the Home Using Simple and Ubiquitous Sensors,” in *Proc. Int. Conf. Pervasive Computing*, Vienna, Austria, 2004.

[20] E. Munguia Tapia, “Activity Recognition in the Home Setting Using Simple and Ubiquitous Sensors,” Master’s thesis, Media Arts and Sciences, Massachusetts Institute of Technology, 2003.

[21] G. Michael Youngblood, “Automating Inhabitant Interactions in Home and Workplace Environments through Data-driven Generation of Hierarchical Partially-observable Markov Decision Processes,” Ph.D. dissertation, The University of Texas at Arlington, Aug. 2005.

[22] T.L.M. van Kasteren, A. K. Noulas, G. Englebienne and B.J.A. Kröse, “Accurate Activity Recognition in a Home Setting,” in *Proc. 10th Int. Conf. Ubiquitous Computing*, Seoul, South Korea, 2008.

[23] T. Reenskaug Website [Internet]. The Model-View-Controller (MVC) - Its Past and Present, University of Oslo; [updated 2003 Aug 20; cited 2010 May 18]. Available from: http://home.ifi.uio.no/trygver/2003/javazone-jaoo/MVC_pattern.pdf.

[24] Averill M. Law, W. David Kelton, *Simulation Modeling and Analysis*, 2nd ed.

[25] Roger S. Pressman, *Software Engineering - A Practitioner’s Approach*, 6th ed.

# BIOGRAPHICAL SKETCH

Shantonu Hossain was born in 1984 in Bangladesh. She completed her Bachelor of Science degree in computer science and engineering in July 2007 from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh. Then she worked as a software engineer in Spectrum Engineering Consortium Ltd., Dhaka for two years. She continued her studies at the University of Florida in the Department of Computer and Information Science and Engineering. She received her Master of Science degree in August 2010. She will be joining the PhD program in the Department of Computer Science at the University of Rochester in September 2010. Her major research interests are pervasive computing, distributed systems and distributed computing.