Gen*: Synthetic Populations Generator for Agent-Based Model

1. Introduction

Gen* is a synthetic population generator for agent-based simulation. It generates synthetic agent populations conforming to input data. Thanks to the GAMA integration, generated synthetic populations can be directly used in GAMA simulations. The development is Gen* is financially supported by the French National Research Agency.

2. Input data

In order to generate synthetic populations, Gen* needs a set of input data. These kinds of data are often available in the public census databases such as the French National Institute of Statistics and Economic Study (INSEE), the United States Census Bureau (USCB), the General Statistique Office of Vietnam (GSO). This section describes some input data supported by Gen*.

2.1. Contingency table

In statistics, a contingency table is a type of table in matrix format that displays the (multivariate) frequency distribution of the variables or attributes. In this text, variable and attribute has the same meaning so they are interchangeable.

For example, suppose that we have two attribute, sex and handedness. Sex has two values: *male* and female. Handedness has two values: *right-handed* and *left-handed*. Further suppose that 100 individuals are randomly sampled from a very large population as part of a study of sex differences in handedness. A contingency table can be created to display the numbers of individuals who are male and right-handed, male and left-handed, female and right-handed, and female and left-handed. Such a contingency table is shown below.

	Right-handed	**Left-handed**	**Total**
Males	43	9	52
Females	44	4	48
Totals	87	13	100

The numbers of the males, females, and right- and left-handed individuals are called **marginal totals**. The **grand total**, i.e., the total number of individuals represented in the contingency table, is the number in the bottom right corner.

The example above is the simplest kind of contingency table, a table in which eachattribute has only two levels (attribute values); this is called a 2 × 2 contingency table. In principle, any number of rows and columns may be used. There may also be more than two attributes. Each attribute also has a data type, e.g., boolean, string, interger, …

For a more detail description of the contingency table, please have a look at [1].

Gen* accepts contingency table as its input data. However, it doesn't expect marginal totals and grand total values in the contingency table. The following paragraphs give some examples of the contingency tables that are used as Gen's input data.

Example 1: Contingency table of sex and age

Table 1 gives the frequency distribution of sex and age group of Bondy town's population. This table includes two concerned attributes: "sex" and "age". The "sex" attribute consists of

two possible values: *Male* and *Female*. The “age” attribute includes the following possible values: *[0, 2], [3, 5], [6, 10], [11, 17], [18, 24], [25, 39], [40, 54], [55, 64], [65, 79], [80, 80+]*

	Men	**Women**	**Total**
< 3 years old	1374	1418	2793
3 to 5 years old	1352	1375	2727
6 to 10 years old	2076	1879	2954
11 to 17 years old	2594	2568	5162
18 to 24 years old	2680	2529	5209
25 to 39 years old	5418	5979	11397
40 to 54 years old	5149	5591	10740
55 to 64 years old	2746	2741	5487
65 to 79 years old	2052	2208	4261
80+ years old	579	1220	1800
Total	26020	27510	53530

Table 1: Frequency distribution of sex and age group

Example 2: Contingency table of household size and social-professional category

Table 2 gives the frequency distribution of the household size and the social-profesionnal category of the household head (an adult is selected as the household head). Two concerned attributes are *household size* and *social-professional category of the household’s head*. The “household size” attribute consists of the following values: *1, 2, 3, 4, 5, 6+*. The “*social-professional category of the household’s head*” includes the following values: *“Farmers”, “Carftsmen, merchants, CEOs”, “Senior managers and intermediate occupations”, “Intermediate professions”, “Employees”, “Retirees”, “Others without professional activity”*.

	Number of people (household size)						
	1	2	3	4	5	6+	Total
Farmers	6	0	0	0	0	0	6
Craftsmen, merchants, CEOs	89	174	194	209	161	94	920
Senior managers and intermediate	402	333	253	270	108	65	1432

occupations							
Intermediate professions	740	658	614	534	219	130	2905
Employees	948	925	758	504	338	185	3659
Workers	621	727	783	903	725	470	4209
Retirees	2133	1856	528	217	110	90	4934
Others without a profesionnal activity	348	265	195	176	76	54	1114
Total	5297	4938	3325	2813	1747	1097	19207

Table 2: Frequency distribution of Household size and social-professional category of household head

Example 3: Contingency distribution of housing's types and housing's number of rooms

Table 3 illustrates the frequency distribution of housing types and housing's number of rooms. Two concerned attributes are "housing type" and "housing's number of rooms". The "housing type" attribute consists of the following attributes: *"Main Housings", "Occasional Housings", "Secondary Housings"* and *"Vacant Housings"*. The "number of rooms" attribute has the following values: *1, 2, 3, 4, 5, 6+*.

	Main Housings	**Occasional Housings**	**Secondary Housings**	**Vacant Housings**	**Total**
1 room	1037	5	4	71	1118
2 rooms	3627	15	12	313	3968
3 rooms	6453	7	32	251	6743
4 rooms	4770	10	23	103	4906
5 rooms	2384	0	3	75	2462
6+ rooms	939	0	3	10	952
Total	19211	38	76	823	20149

Table 3: Frequency disitrbution of housing types and housing's number of rooms

2.2. Inference/deduction relationship between two attributes

This data type establishes a relationship between the values of two attributes. If we know the value of one attribute, we can infer the value of other attribute. The following table gives the inference/deduction relationship between two attributes: Social-Professional Category and

Hourly Net Salary. According to the relationship, if the value of “Social-Professional Category” is “Farmers”, we can deduce that the value of “Hourly Net Salary” is [5, 10].

Social-Professional Category	Hourly Net Salary
Farmers	[5, 10]
Craftsmen, merchants, CEOs	[8, 20]
Managers and higher intellectual professions	[10, 18]
Intermediate professions	[10, 18]
Employees	[7, 14]
Workers	[6, 12]
Retirees	[4, 11]
Others without professional activity	[2, 7]

Table 4: Inference/deduction relationship between “social-professional category” and “hourly net salary” attributes.

2.3. Sample data

Sample data is a set of individuals randomly sampled from a population. To collect the sample data, we often perform surveys on the target population. For example, the following table gives a sample data of a population of people of a city. The sample shows that we are interested in two attributes: *Gender* and *Age*. The sample size is 1000 people.

No	Gender	Age
1	Male	35
2	Female	37
3	Female	60
...	...	...
1000	Male	40

Table 5: A sample data of a population.

Gen* offers some GAML operators to transform sample data into contignecy table (frequency distribution of attributes) which then can be fed directly to the generator as input data.

3. Meta-model

Gen* bases on a meta-model which defines concepts for representing the input data. This section describes the Gen*'s meta-model.

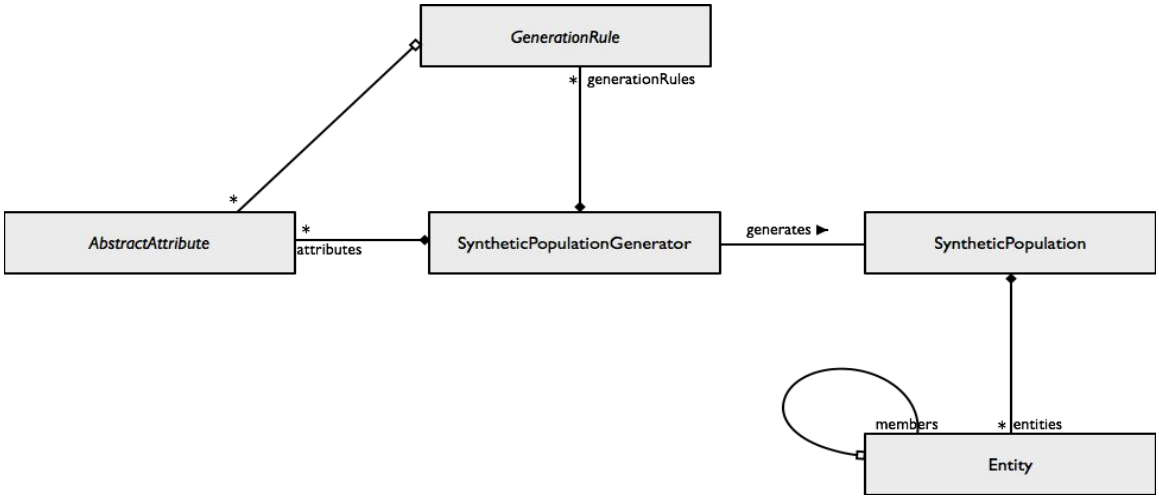

Figure 1: Meta-model (part 1)

Figure 1 depicts the first part of the meta-model. In order to generate a synthetic population, we define a SyntheticPopulationGenerator. A SyntheticPopulationGenerator consists of one or several AbstractAttributes and GenerationRules.

An AbstractAttribute represents an attribute of the entity that we would like to generate, e.g., age, sex, socio-professional category.

A GenerationRule (a generation rule) encapsulates an input data format, for example, contingency table (frequency distribution), inference/deduction relationship. It defines how an input data is used to generate the attribute values.

A generator, SyntheticPopulationGenerator, is therefore based on a set of generation rules for generating a synthetic population.

A SyntheticPopulation represents a synthetic population generated by a SyntheticPopulationGenerator. It consists of a set of generated entities. For example, a synthetic population of inhabitants contains a set of generated inhabitants. A synthetic population of households includes a set of generated households.

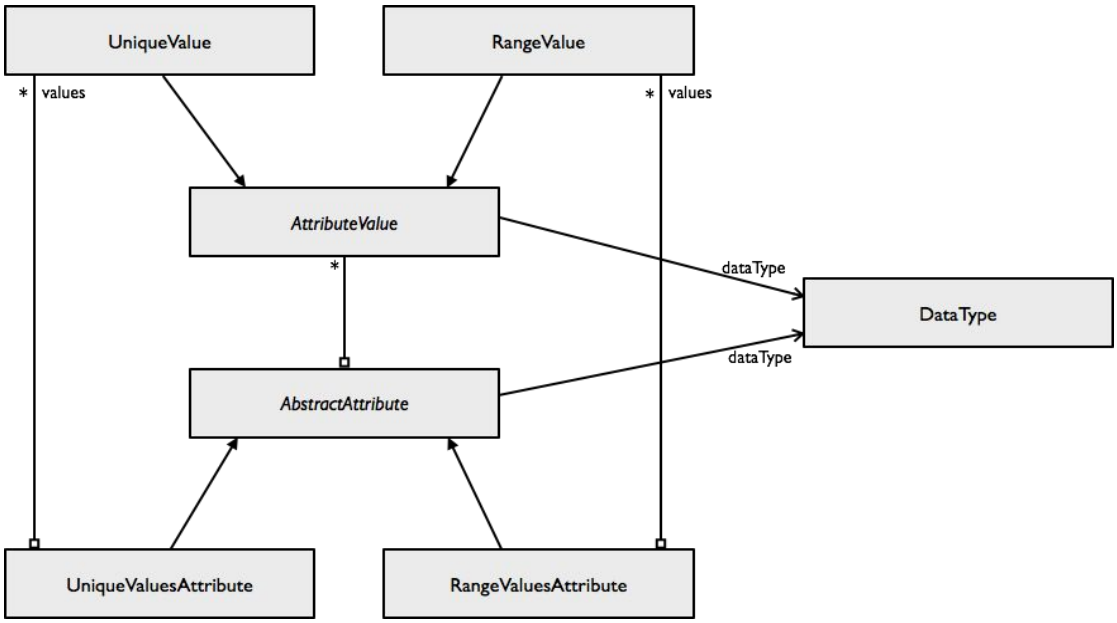

Figure 2: Meta-model (part 2)

Figure 2 presents two attribute types supported by Gen*: UniqueValuesAttribute and RangeValuesAttribute. There are two types of attribute value: UniqueValue and RangeValue. UniqueValue represents the unique values, e.g., male, female, 1, 2, … RangeValue encapsulates range values, e.g., [0, 2], [3, 5]. An AttributeValue has a data type such as boolean, string, integer, … The DataType concept represents the data type of an AttributeValue.

An UniqueValuesAttribute can consist of several instances of UniqueValue. For example, “sex” attribute of “inhabitant” population can be represented as an instance of UniqueValuesAttribute which consists of two instances of UniqueValue. These instances has boolean data type and take “true” and “false” as values to represent “male” and “female”.
“Number of rooms” attribute of household can be represented as an instance of UniquesValueAttribute. This instance contains six instances of UniqueValue having integer as data type to represent six integers: 1, 2, 3, 4, 5, 6.

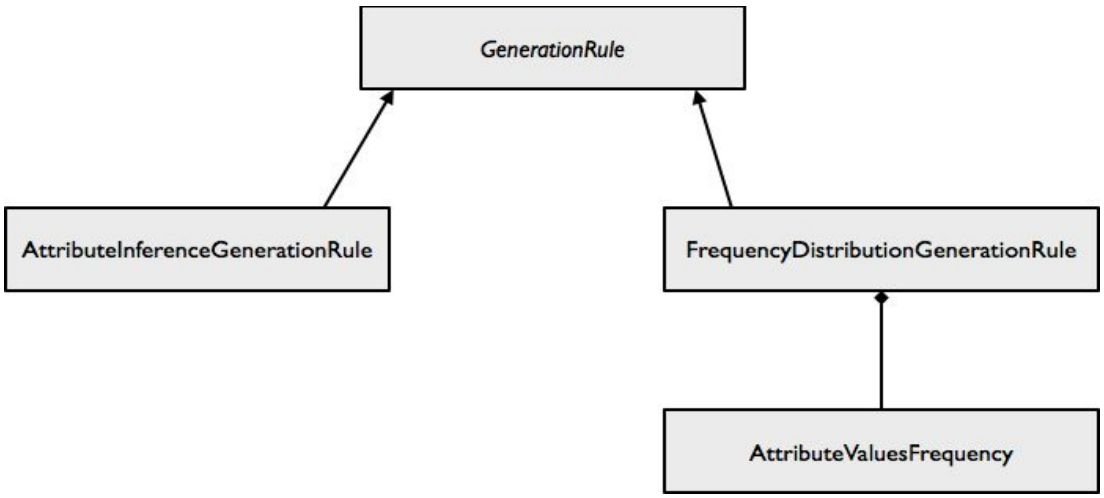

Figure 3: Meta-model (part 3)

Figure 3 depicts two types of generation rule supported by Gen*: FrequencyDistributionRule and AttributeInferenceGenerationRule. A FrequencyDistributionRule represents a contingency table (e.g., table 1, table 2, table 3). An AttributeInferenceGenerationRule encapsulates an inference/deduction relationship between two attributes (e.g., table 4). The AttributeValuesFrequency represents the number of entities the same set of attribute values (frequency). For example, consider the population of Bondy by sex and age (table 1), the number of entities with {sex = male, age = [0, 2]} is 1374. The number of entities with {sex = female, age = [3, 5]} is 1375.

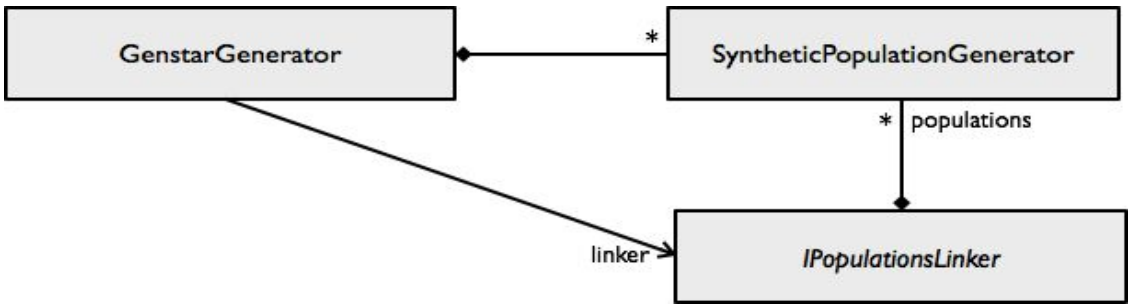

Figure 4: Meta-model (part 4)

GenstarGenerator represents the generation of synthetic populations. GenstarGenerator is composed of one or several SyntheticPopulationGenerator. For example, a GenstarGenerator consists of two synthetic population generators (SyntheticPopulationGenerator): “generator 1” and “generator 2”. Generator 1 is responsible for generating the population of households. Generator 2 is in charge of generating the population of inhabitants.
IPopulationLinker helps to establish the relationships between the entities generated by the synthetic population generators. For example, a IPopulationLinker can be responsible for creating the compositional relationship between “household” and “inhabitant” entities, i.e., a household consists of one or several inhabitants as members.

4. GAMA integration

Gen* is integrated with GAMA platform. The modeller can control the generator in the GAMA model thanks to the provided GAML operators. For example, the modeler can use GAML operators to generate synthetic populations which can be used to create agents for the GAMA model. This section presents how we can generate synthetic populations using GAML operators.

Generating a synthetic population

In order to generate a synthetic population, the first step is to prepare input data. The second step is to use GAML operators to generate the synthetic population from the input data. This section details these two steps.

Input data

Input data for the generation of a synthetic population are formulated as the following CSV files:

1. A CSV file defining the attributes that we want to generate
2. A CSV file listing the generation rule files
3. One or several CSV files defining the generation rule(s)

Comma (,) is used as the delimiter between elements of the CSV files.

Attributes

Attributes of the population that we would like to generate are defined in a CSV file. The first line is the file header. Each of the following lines defines an attribute. The following table describes the file header.

Order	Content	Signification
1	Name On Data	attribute name on input data
2	Name On Entity	attribute name on generated entity. An attribute with the same name is declared on the (generated) GAMA agent/species.
3	Data Type	attribute data type
4	Value Type On Data	attribute value type on input data. There are two attribute value types: Unique (for UniqueAttributeValue) and Range (for RangeAttributeValue). A range value consists of a min value and a max value which are delimited by a colon (min_value:max_value).
5	Values	a list of attribute values. Each attribute value is delimited by a semicolon (;). Attribute values type is defined by “Value Type On Data”.
6	Value Type On Entity	attribute value type on generated entity. There are also two attribute value types (Unique and Range) as attribute value type on input data.

Table 6: Header of the attribute file.

The following table gives an example of an attribute file.

```
Name On Data,Name On Entity,Data Type,Value Type On Data,Values,Value Type On Entity
Category,category,string,Unique,C0; C1; C2; C3; C4; C5; C6; C7,Unique
Gender,gender,bool,Unique,true; false,Unique
Age,age,int,Range,0:4; 5:17; 18:24; 25:34; 35:49; 50:64; 65:100,Unique
```

Table 7: Attribute file example 1

The first line is the file header.
The second line is the declaration of the “Category” attribute. “category” is the attribute name on GAMA agent. The data type is string. Value types on data and on entity are unique values. Possible attribute values include 8 values (C0 -> C7).
The third line is the declaration of the “Gender” attribute. “gender” is the attribute name on GAMA agent. This is a boolean attribute with unique value type on both input data and entity. Two possible values are true and false.
The fourth line is the declaration of the “Age” attribute. “age” is the attribute name on GAMA agent. This is an integer attribute. Value type on data is range. Value type on GAMA agent attribute is unique. Possible values are 0:4; 5:17; 18:24; 25:34; 35:49; 50:64; 65:100. To generate a unique value from a range value, Gen* selects a random unique value within the generated range value. For example, the generated range value (from the possible values) is 18:24. The randomly selected unique value can be 20.

For the sake of clarity in this document, from now on, the comma delimiter won’t appear in the CSV file anymore. The CSV file will be presented in the following tabular format.

Name On Data	Name On Entity	Data Type	Value Type On Data	Values	Value Type On Entity
Category	category	string	Unique	C0; C1; C2; C3; C4; C5; C6; C7	Unique
Gender	gender	bool	Unique	true; false	Unique
Age	age	int	Range	0:4; 5:17; 18:24; 25:34; 35:49; 50:64; 65:100	Unique

Table 8: Attribute file showed in tabular format.

Generation rules

The generation rules used by the generator are listed in CSV file. The first line is the file header. Each of the following lines points to a generation specification file. The following table gives an example of the generation rule list file.

Name	File	Rule Type
Distribution 1	../includes/population/People_GenerationRule1_Data.csv	Frequency Distribution
Distribution 2	../includes/population/People_GenerationRule2_Data.csv	Frequency Distribution

Table 9: Example of Generation rule list

The signification of the file header is detailed in the following table.

Header content	Signification
Name	name of the generation rule
File	path to the generation rule specification file. In a GAMA model

	project, this is often the relative path (with respect to the location of the model file) to the generation rule specification file.
Rule Type	type of the generation rule. It can be either Frequency Distribution or Inference

Table 10: Header of generation rule list file.

Generation rule specification

This CSV file gives the detail specification of a generation rule. The following two tables give examples of the frequency distribution generation rule.

Category:Output	Age:Output	Frequency
C0	0:4	0
C1	0:4	0
C2	0:4	0
C3	0:4	0
C4	0:4	0
C5	50:64	0
C6	50:64	2
C7	50:64	0
...	...	...
C4	65:100	1
C5	65:100	2123
C6	65:100	0
C7	65:100	0

Table 11: Frequency distribution generation rule example 1.

Category:Input	Gender:Output	Frequency
C0	TRUE	2221
C1	TRUE	203
C2	TRUE	301
...	...	...
C5	FALSE	1096
C6	FALSE	1407
C7	FALSE	0

Table 12: Frequency distribution generation rule example 2.

The first line is the file header. The last element (Frequency) of the header is the distribution frequency. Other elements of the header are written under the following format: “attribute name”:”attribute type”. “attribute name” is the attribute name. “attribute type” can take either Input or Output as value. The “Input” attribute type means that Gen* reads the attribute value from the GAMA agent. The “Output” attribute type means that Gen* generates the attribute value then writes the value to the GAMA agent.

GAML operators

In order to generate a synthetic population, the modeler can use “population_from_csv” operator. This operator expects the following parameters:

1. Name of a CSV file defining the attributes of the entity that we want to generate
2. Name of a CSV file listing the generation rule files
3. The number of entities that we would like to generate

The operator returns a list of maps that can be used with the “create” statement to create agents. The GAML pattern for combining “population_from_csv” operator with “create” statement is as follows:

```
list generated_synthetic_population <- population_from_csv(attribute_file.csv, generation_rules.csv, number_of_entities);
    create species_name from: generated_synthetic_population;
```

Table 13: Pattern for creating GAMA agents from a generated synthetic population.

According to table 13, we see that, firstly, we use the “population_from_csv” operator to ask Gen* to generate a synthetic population. Then, secondly, we create GAMA agents from the generated population.

Establishing the compositional relationship between agents

In many cases, we need to generate several synthetic populations that have the compositional relationship between agents of different populations. For example, we would like to generate two synthetic populations: a synthetic population of households and a synthetic population of people. There is a compositional relationship between agents of these populations in the sense that several “people” agents live in a “household” agent. Therefore, we need to establish the relationship between the generated agents. Gen* supports the modeler to create this relationship thanks to the *IPopulationsLinker* concept. This concept is realized in GAMA as the *IGamaPopulationsLinker* (Java) interface. To establish the relationship between generated agents of different populations, we need to follow the following steps:

1. Implement an *IGamaPopulationsLinker* in Java (i.e., write a class that implements the *IGamaPopulationsLinker* interface) which is responsible for establishing the compositional relationship between agents.
2. Generate the synthetic populations then create GAMA agents from the synthetic populations
3. Use the “link_population” operator to ask the *IGamaPopulationsLinker* (implemented in step 1) to establish the relationship between agents.

The GAML pattern for creating the relationship between agents of different populations is as follows:

```
    list household_population <- population_from_csv('../Household_Attributes.csv', '../Household_GenerationRules.csv', nb_of_households);
    create household from: household_population returns: generated_households;
```

```
        list people_population <- population_from_csv('../People_Attributes.csv',
'../People_GenerationRules.csv', nb_of_people);
        create people from: people_population returns: generated_people;

        list<list> populations_to_link <- [ generated_households, generated_people ];
        let linker_result <- link_populations('household_size_linker', populations_to_link);
```

Table 14: Pattern for establishing the relationships between agents of different synthetic populations.

According to table 14, we are dealing with two populations: “household” and “people”. First of all, we generate the population of “household” and create “household” agents from the generated population. Then we generate the population of “people” and create “people” agents from the generated population. After that we use the “link_populations” operator to ask the “household_size_linker” population linker to establish the compositional relationship between “household” and “people” agents. This linker puts people into a house according to the “household_size” attribute of the “household” agent.

The following table gives the source code of the IGamaPopulationsLinker interface. The “establishRelationship” method is responsible for creating the compositional relationship between agents of different populations.

```
public interface IGamaPopulationsLinker {

    public abstract void setTotalRound(final int totalRound);

    public abstract int getTotalRound();

    public abstract int getCurrentRound();

    public abstract void establishRelationship(final IScope scope, final IList<IList<IMacroAgent>>
populations);
}
```

Table 15: IGamaPopulationsLinker interface.

To develop a linker, we write a Java class that implements the *IGamaPopulationsLinker* interface and give the linker a name. The following table gives the code skeleton of the “household_size_linker” population linker. The “establishRelationship” method is responsible for putting people into households according to household size. The linker name can then be refered in the GAML code if we want to execute the linker (e.g., see table 14).

```
@populations_linker(name = "household_size_linker")
public class MiroPopulationsLinker implements IGamaPopulationsLinker {

    @Override
    public void setTotalRound(int totalRound) {
    }

    @Override
    public int getTotalRound() {
```

```
        return 0;
    }

    @Override
    public int getCurrentRound() {
        return 0;
    }

    @Override
    public void establishRelationship(final IScope scope, final IList<IList<IMacroAgent>> populations) {
        // putting people into households according to household's size
    }
}
```

Table 16: Code skeleton of "household_size_linker" population linker.

5. Case studies

5.1. MIRO model

In MIRO (urban mobility) model, we need to generate the population of residents and households in Grenoble city. We would like to generation two synthetic populations: "people" and "household". A household consists of several people. For "people" entity, we need to generate the following attributes: social-professional category, age and sex. Concerning "household" entity, we would to generate "household size" attribute.

Generating "people" population

Attributes

Generated attributes of "people" population are defined in the following CSV file

Name On Data	Name On Entity	Data Type	Value Type On Data	Values	Value Type On Entity
Category	category	string	Unique	C0; C1; C2; C3; C4; C5; C6; C7	Unique
Gender	gender	bool	Unique	true; false	Unique
Age	age	int	Range	0:4; 5:17; 18:24; 25:34; 35:49; 50:64; 65:100	Unique

Table 17: People_Attributes.csv

Generation rules

The list of generation rules is defined in People_GenerationRules.csv file. We can see that this generator consists of two generation rules.

Name	File	Rule Type
Distribution 1	../includes/population/People_GenerationRule1_Data.csv	Frequency Distribution

Distribution 2	../includes/population/People_GenerationRule2_Data.csv	Frequency Distribution

Table 18: People_GenerationRules.csv

Generation rule specification

The specification of our two generation rules is found in the following two CSV files. The first CSV file details the frequency distribution of social-professional category and age attributes. The second CSV file is the frequency distribution of social-professional category and gender attributes.

Category:Output	Age:Output	Frequency
C0	0:4	0
C1	0:4	0
C2	0:4	0
C3	0:4	0
C4	0:4	0
C5	0:4	0
C6	0:4	0
C7	0:4	0
C0	5:17	0
C1	5:17	7
C2	5:17	6
C3	5:17	0
C4	5:17	14
C5	5:17	0
C6	5:17	2876
C7	5:17	0
C0	18:24	0
C1	18:24	76
C2	18:24	102
C3	18:24	0
C4	18:24	1136
C5	18:24	0
C6	18:24	0
C7	18:24	0
C0	25:34	1253
C1	25:34	218
C2	25:34	144
C3	25:34	0
C4	25:34	97
C5	25:34	3
C6	25:34	9

C7	25:34	0
C0	35:49	2658
C1	35:49	630
C2	35:49	195
C3	35:49	0
C4	35:49	19
C5	35:49	5
C6	35:49	10
C7	35:49	0
C0	50:64	0
C1	50:64	371
C2	50:64	129
C3	50:64	2706
C4	50:64	16
C5	50:64	0
C6	50:64	2
C7	50:64	0
C0	65:100	3
C1	65:100	11
C2	65:100	1
C3	65:100	0
C4	65:100	1
C5	65:100	2123
C6	65:100	0
C7	65:100	0

Table 19: People_GeneratiionRule1.csv

Category:Input	**Gender:Output**	**Frequency**
C0	TRUE	2221
C1	TRUE	203
C2	TRUE	301
C3	TRUE	1458
C4	TRUE	649
C5	TRUE	1035
C6	TRUE	1490
C7	TRUE	0
C0	FALSE	1693
C1	FALSE	1110
C2	FALSE	276
C3	FALSE	1248
C4	FALSE	634

C5	FALSE	1096
C6	FALSE	1407
C7	FALSE	0

Table 20: People_GenerationRule2.csv

Generating “household” population

Attributes

We need to generate only one attribute of the “household” population: Household Size. This attribute represents the number of people in a household.

Name On Data	Name On Entity	Data Type	Value Type On Data	Values	Value Type On Entity
Household Size	householdSize	int	Unique	1; 2; 3; 4; 5; 6; 7; 9	Unique

Table 21: Household_Attributes.csv

Generation rules

The “household” generator has one generation rule.

Name	File	Rule Type
Distribution 1	../includes/population/Household_GenerationRule_Data.csv	Frequency Distribution

Table 22: Household_GenerationRules.csv

Generation rule specification

The following CSV file details the specification of the generation rule.

Household size:Output	Frequency
1	1957
2	2320
3	856
4	877
5	333
6	62
7	11
8	4
9	0

Table 23: Household_GenerationRule_Data.csv

Establishing the compositional relationship between “people” and “household”

To establish the compositional relationship between agents “people” and “household” population, we develop a Java class that implements the IGamaPopulationsLinker. We name our linker “miro_population_linker”.

```
package ummisco.genstar.gama.populations_linker;

import java.util.List;

import msi.gama.metamodel.agent.IMacroAgent;
import msi.gama.precompiler.GamlAnnotations.populations_linker;
import msi.gama.runtime.IScope;
import msi.gama.runtime.exceptions.GamaRuntimeException;
import msi.gama.util.GamaListFactory;
import msi.gama.util.IList;
import msi.gaml.extensions.genstar.IGamaPopulationsLinker;
import msi.gaml.types.Types;


@populations_linker(name = "miro_household_size_linker")
public class MiroPopulationsLinker implements IGamaPopulationsLinker {

    @Override public void setTotalRound(int totalRound) { }

    @Override public int getTotalRound() { return 0; }

    @Override public int getCurrentRound() { return 0; }

    @Override public void establishRelationship(final IScope scope, final IList<IList<IMacroAgent>>
populations) {
        if (populations.size() != 2) { GamaRuntimeException.error("'populations' must contain 2
populations.", scope); }

        List<IMacroAgent> firstPopulation = populations.get(0);
        List<IMacroAgent> secondPopulation = populations.get(1);

        if (firstPopulation.isEmpty() || secondPopulation.isEmpty()) { return; }

        String firstSpeciesName = firstPopulation.get(0).getSpeciesName();
        String secondSpeciesName = secondPopulation.get(0).getSpeciesName();

        String GAML_PEOPLE_SPECIES = "people";
        String GAML_HOUSEHOLD_SPECIES = "household";

        if ( ( !firstSpeciesName.equals(GAML_PEOPLE_SPECIES) &&
!firstSpeciesName.equals(GAML_HOUSEHOLD_SPECIES) )
|| (!secondSpeciesName.equals(GAML_PEOPLE_SPECIES) &&
!secondSpeciesName.equals(GAML_HOUSEHOLD_SPECIES)) ) {
            GamaRuntimeException.error("Invalid GAML species. Expected [" +
GAML_PEOPLE_SPECIES + ", " + GAML_HOUSEHOLD_SPECIES + "] species.", scope);
        }

        List<IMacroAgent> peoplePopulation;
        List<IMacroAgent> householdPopulation;
        if (firstSpeciesName.equals(GAML_PEOPLE_SPECIES)) {
```

```
            peoplePopulation = firstPopulation;
            householdPopulation = secondPopulation;
        } else {
            peoplePopulation = secondPopulation;
            householdPopulation = firstPopulation;
        }

        int memberAttributeSet = 0;
        int peoplePopulationIndex = 0;
        for (IMacroAgent household : householdPopulation) {
           Integer householdSize = (Integer) household.getAttribute("householdSize");

            // GAML field name: "member_people"
            IList<IMacroAgent> member_people = GamaListFactory.create(Types.AGENT);

            for (int peopleCurrentIndex = peoplePopulationIndex; peopleCurrentIndex <
(peoplePopulationIndex + householdSize);  peopleCurrentIndex++) {
                if (peopleCurrentIndex >= peoplePopulation.size()) { return; } // out of people
                   member_people.add(peoplePopulation.get(peopleCurrentIndex));
                }
                peoplePopulationIndex += member_people.size();

                household.setAttribute("member_people", member_people);
                memberAttributeSet++;
            }
    }
}
```

Table 24: Java implementation of “miro_household_size_linker”.

The implementation of the populations linker is a little bit long. However, we are interested in the “establishRelationship” method. The main idea of this method is that it iterates through two GAMA agent populations: “people” and “household” then puts people agents into household agents according to the “householdSize” attribute. The linker tries to ensure that the number of people of a household is equal to “householdSize” attribute value of the corresponding household. There may exist some household agents that don’t have any member people although the “householdSize” attribute is positive. The reason is that there is no more available people (people that are not yet belong to any household). Therefore, the linker can not assign members for these households.

To invoke this populations linker in GAML, we can use the “link_populations” operation and refer to the linker name. The following GAML sample model shows how to generate two synthetic populations (“people” and “household”) then ask the “miro_household_size_linker” population to establish the compositional relationship between “people” and “household” agents.

```
/**
 *  Author: voducan
 *  Description:
 */

model household_with_people
```

```
global {

	int nb_of_people <- 14821;
	int nb_of_households <- 6420;

	init {
		list miro_household_population <-
population_from_csv('../includes/population/Household_Attributes.csv',
'../includes/population/Household_GenerationRules.csv', nb_of_households);
		create household from: miro_household_population returns: generated_households;

		list miro_people_population <-
population_from_csv('../includes/population/People_Attributes.csv',
'../includes/population/People_GenerationRules.csv', nb_of_people);
		create people from: miro_people_population returns: generated_people;

		list<list> populations_to_link <- [ generated_households, generated_people ];
		let linker_result <- link_populations('miro_household_size_linker',
populations_to_link);


		// TODO: open the "household" population browser and have a look at the "member"
list to verify the linker result

	}
}

species people {

	string category;
	bool gender;
	int age;


	aspect base {
		draw circle(2) color: #yellow;
	}
}

species household {
	int householdSize;
	list<people> member_people;
}

experiment miro_household_with_people type: gui {
	output {

	}
}
```

Table 25: Sample model of populations linker usage.

After loading the “miro_household_with_people” experiment in GAMA, we can inspect “household” agents to see that “member_people” list contains “people” agents as members.

References

[1] Contingency table https://en.wikipedia.org/wiki/Contingency_table