A Modeling Method for Representation of Geographical Information of a System-of-Systems

Young-Min Baek, Eunho Cho, Yong-Jun Shin, and Doo-Hwan Bae
School of Computing
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon, Republic of Korea
{ymbaek, ehcho, yjshin, bae}@se.kaist.ac.kr

2021 16th International Conference of System of Systems Engineering (SoSE) | 978-1-6654-4454-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SOSE52739.2021.9497462

***Abstract*—Due to the increasing interest in Systems-of-Systems (SoSs) and SoS engineering (SoSE), a considerable number of model-based SoSE methods have been actively researched in recent years. Since a constituent of an SoS may not be originally designed to constitute a higher-level organization or SoS, the integration of distributed independent constituents is a crucial SoSE activity. However, the distribution poses challenges for SoSE because this characteristic inevitably increases uncertainties that can largely influence constituents' behaviors and interactions. Therefore, this uncertain distribution should be systematically modeled and simulated and a geographical map (GeoMap) should be appropriately embedded in a simulation model as a fundamental simulation object. However, existing simulation techniques do not concentrate on geographical features; they only focus on the structural and behavioral aspects of models. To support the modeling and simulation of the geographical features of an SoS, this study proposes a modeling method that flexibly represents geographical information. On the basis of SoS-specific characteristics, our method provides a scalable modeling technique for building, updating, and managing a GeoMap for simulation. An open-sourced project, called *GeoMapBuilder-for-SoS*, is developed to simulate the geographically distributed entities of an SoS and demonstrate the applicability and feasibility of the proposed modeling method.**

***Keywords*—Model-based System-of-Systems Engineering, Geographical Map, Distributed System, Modeling & Simulation**

I. Introduction

In recent years, a considerable number of large-scale systems have been engineered to integrate autonomous systems and achieve higher-level common goals. For this reason, since the 2000s, Systems-of-Systems (SoSs) and SoS engineering (SoSE) have been actively studied in both academia and various industrial domains [1], [2]. An SoS comprises multiple distributed constituents that can have managerial and operational independence [3]. A constituent of an SoS might not be originally designed as a part of the SoS; thus, the integration, and orchestration of diverse constituents are the most crucial goals of SoSE. To systematically integrate such constituents into SoSs, which may have large sizes and high complexity, many model-based approaches have been studied for SoSE, such as model-based requirements engineering, design, testing, and verification [4], [5].

For effective model-based SoSE (MBSoSE), an SoS should be properly modeled from various angles and aspects for different purposes of engineering activities. A study by Nielsen *et al.* [2] defined major dimensions for positioning an MBSoSE approach; the authors stated that one of the dimensions that SoS engineers should consider is the distribution of constituents. Their physical and virtual distributions inevitably increase uncertainties that influence constituents' states and behaviors while interacting with other dispersed entities. Proper infrastructure and communication media for MBSoSE cannot be designed and developed without consideration of the geographical distribution and dispersion of constituents.

Effective analysis of an SoS requires a modeling and simulation (M&S) technique for observing and evaluating the possible behaviors of an SoS before releasing or deploying the SoS [6]. Even though M&S processes can vary according to the type of SoS and its objectives, geographical features and information need to be represented in the simulation model appropriately. Simulation results can be used for critical decision making, such as cost-benefit analysis and safety and risk analysis, to address behavioral and structural uncertainty issues caused by geographical changes. However, existing SoS simulation approaches mainly focus on the structural and behavioral aspects of SoSs; they do not consider geographical characteristics as the first tier entities of SoSs. Even though domain-specific map modeling approaches have been developed, such as those used in traffic map modeling, a forest map modeling, and game level design, they only leverage fixed sets of domain-specific geographical features; thus these approaches are not scalable for extensions, updates, and modifications.

For the same purpose, related studies have developed domain-specific geography modeling methods that build maps that mainly focus on the representation of the location points of objects. Some markup languages were developed to represent geographical entities and objects using basic geometric classes, such as coordinates (e.g., xPos, yPos, longitude, latitude, and altitude), lines, and polygons [7], [8]. They provide well-established formalisms and interfaces to build general-purpose maps via visualization means, but they share common limitations from the lack of a conceptual framework. Without such conceptualization, a technique only can provide a fixed set of geographical semantics; thus, it cannot support a scalable method to flexibly model different types of maps according to evolving requirements and different domains.

To address these issues in the M&S of the geographical features of SoSs, this study proposes a method of modeling the geographical information of an SoS. The contributions of this paper can be summarized as follows.

- This study formally defines a geographical map (GeoMap) and the properties of an SoS by conceptualizing the general geographical information of the SoS.
- On the basis of the conceptualization, this study proposes a scalable method that enables the concise modeling of a GeoMap to manage the geo-locations of SoS objects.
- The feasibility of our approach is demonstrated by applying the method to an actual simulation engine and modeling several types of maps possibly used in SoS domains.

II. Related Work

Some previous studies developed approaches that represent the geographical properties of software and systems. One of the mainstreams in this topic is the markup language-based modeling approach, which provides geometric and geographical semantics, syntax, and corresponding notations. *Geography Markup Language* (GML) is an XML-based markup language, and a standardized method of modeling geographical data [7]. GML supports modeling physical entities, such as buildings and persons, as features; geometric properties can be expressed as GML features in a formal way, such as points, lines, and coordinate-based polygons. There are application schemas that are based on the basic GML definition, such as *CityGML* [9], which extends the GML to a specific domain. As another markup language, *Keyhole Markup Language* (KML) was developed to model and visualize graphical data on *Google Earth* and *Google Maps* [8]. To locate an object on these maps, one locates a KML object using the longitude, latitude, and altitude, and this object can be visualized and animated also.

A UML-based modeling method was proposed to represent the geographical properties of a system by extending the UML profile, including stereotypes, tags, and constraints. *GeoFrame* [10] and *GeoProfile* [11] utilize the UML extension so that a modeling engineer specifies geographical objects and properties using the UML syntax. Various modeling languages have been developed for existing methods, but these languages only utilize general geographical properties, such as coordinates (e.g., longitude, latitude, and altitude). Moreover, they were developed without a general conceptual framework; thus, building maps may require much effort to flexibly react/adapt to geographical changes and events.

Several simulation engines that represent geographical maps (GeoMaps) have been developed. *Simulation of Urban MObility* (SUMO) is an open-source simulation tool for traffic networks [12] that simulates a traffic environment with vehicle agents and a routing algorithm. SUMO obtains a traffic network map written in XML as an input, which can be converted from GML files. SUMO utilizes traffic domain-specific models and data to perform its simulation. *AnyLogic* is another popular simulation software for various domains, such as traffic networks, physical plants, and warehouse operations [13]. Using AnyLogic, an engineer can model and

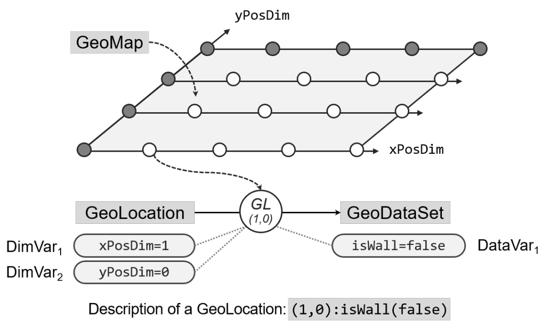

Fig. 1. Conceptualization of a GeoMap

simulate a geographically-distributed system, such as a traffic network, based on a two-dimensional (2D) space. Although AnyLogic supports the user-defined or library-based definition of a map, it does not support scalable extensions of geographical features.

III. Definitions based on Conceptualization

A multi-dimensional geographical map (GeoMap) can be conceptualized as depicted in Figure 1. The map in the figure is a two-dimensional *GeoMap* that is described by two *dimension variables* (*GeoDimensions*), which are `xPosDim` and `yPosDim`. If the dimension variables have concrete values (e.g., `xPosDim==1`), then the map can return a specific location point, called *GeoLocation*. Each GeoLocation has an individual set of data, called *GeoData*, and it describes the spatial features of interest. In the example, the GeoData is the `isWall` variable, which indicates whether a GeoLocation in the 2D plane is a wall.

The concepts of *GeoMap*, *GeoLocation*, *GeoDimension*, and *GeoData* are defined as below, and formal representations are shown for each concept.

- **GeoMap**: A $GeoMap$, which stands for a Geographical Map, is a model that represents a particular (physical or logical) area/space to show geographical features and data. A GeoMap consists of multiple unit location points, called *GeoLocations*. A GeoMap is defined as a function from a *GeoLocation* to a *GeoDataSet* to represent the information of whole location points, . The *GeoLocation* aggregates a set of *GeoDimension* variables, and the *GeoDataSet* represents a set of *GeoData* variables. Since the dimensions and data are logically designed to represent discrete values, a GeoMap has a finite number of locations.

$$GeoMap : GeoLocation \rightarrow GeoDataSet$$

A semi-formal definition and an example of a *GeoMap* definition are presented as follows:

```
defMap map_name:(GeoDimension [,GeoDimension])
                         -> (GeoData [,GeoData])
defMap MapExample:(xPosDim,yPosDim)->(isWall)
```

- **GeoLocation**: A $GeoLocation$ is a geographical unit that describes a particular location point on a GeoMap. A GeoLocation is pinned and identified by a set of conditions of the GeoMap's GeoDimensions, and the condition

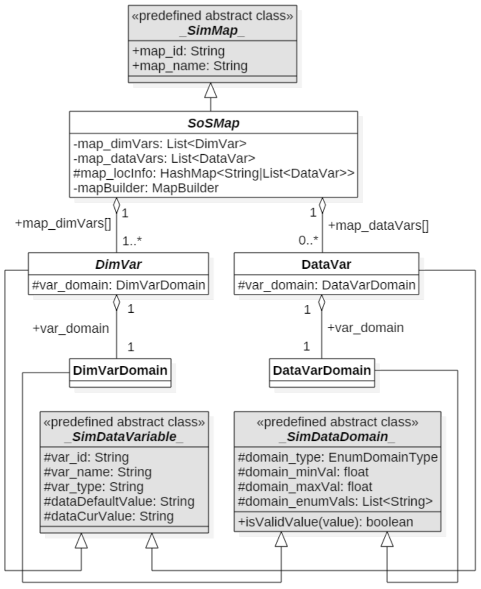
«predefined abstract class»
SimMap
+map_id: String
+map_name: String
SoSMap
-map_dimVars: List<DimVar>
-map_dataVars: List<DataVar>
#map_locInfo: HashMap<String|List<DataVar>>
-mapBuilder: MapBuilder
+map_dimVars[]
1
1..*
1
0..*
+map_dataVars[]
DimVar
#var_domain: DimVarDomain
DataVar
#var_domain: DataVarDomain
+var_domain
+var_domain
DimVarDomain
DataVarDomain
«predefined abstract class»
SimDataVariable
#var_id: String
#var_name: String
#var_type: String
#dataDefaultValue: String
#dataCurValue: String
«predefined abstract class»
SimDataDomain
#domain_type: EnumDomainType
#domain_minVal: float
#domain_maxVal: float
#domain_enumVals: List<String>
+isValidValue(value): boolean

Fig. 2. A GeoMap Meta-Model

is described by a conditional expression of the variable name and value. Therefore, a *GeoLocation* is defined as a Cartesian product of one or more *GeoDimensions* based on their value domains.

$$GeoLocation = \prod_{n} DimVar_n$$

where n is the number of *DimVars* and $n > 0$. To specify a particular location, a *GeoLocation* should be able to represent conditional expressions (*dimVarCond*) on the basis of the dimensions. The following is an example of a *GeoLocation* (1,0) specification:

```
(xPosDim==1&&yPosDim==0)
```

– **DimVar**: A $DimVar$ is a variable used to describe a particular dimension of a GeoMap and constrained by a concrete domain. Here, a variable commonly includes its default value (*defval*) and current value (*curval*).

$$DimVar = (defval, curval, VarDomain)$$

A domain of a variable is semi-formally represented by the values below in square brackets for the definition of a dimension variable. The following example shows the definition of an xPos variable without the *defval* and *curval*.

```
defVar xPosDimVar( , ,[0,9])
```

- **GeoDataSet**: A $GeoDataSet$ is a set of *DataVars* for all the *GeoLocations* of a GeoMap, and all the points of the map commonly include a set of GeoData with specific values. Similar to the GeoLocation, GeoDataSet is defined as a product of zero (∅) or more *DataVars*.

$$GeoDataSet = \emptyset \text{ or } \prod_{n} DataVar_n$$

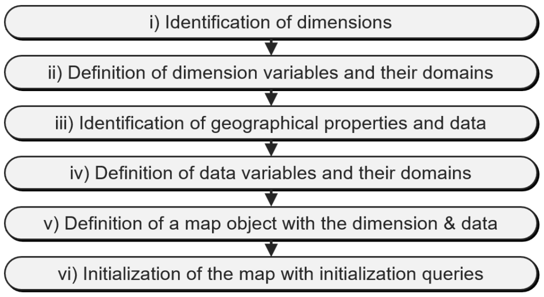
i) Identification of dimensions
ii) Definition of dimension variables and their domains
iii) Identification of geographical properties and data
iv) Definition of data variables and their domains
v) Definition of a map object with the dimension & data
vi) Initialization of the map with initialization queries

Fig. 3. Overall Process to Develop a GeoMap

– **DataVar**: A $DataVar$ is a variable used to include the data/information of a GeoLocation of a GeoMap.

$$DataVar = (defval, curval, VarDomain)$$

Simliar to a *DimVar*, a *DataVar* which has `TRUE` as a defVal and no curVal is semi-formally defined as follows.

```
defVar isWallVar(FALSE, ,[TRUE,FALSE])
```

- **VarDomain**: A $VarDomain$ constrains and specifies the range of possible values of a variable, such as *DimVar* and *DataVar*.A domain for numerical variables specifies the minimum and maximum values (e.g., [0,100]), and a domain for enumeration variables should specify a list of allowed data values (e.g., [“FLOORB1”, “FLOOR1”, “FLOOR2”]).

IV. Model-based Development of a GeoMap

A. GeoMap Metamodel

For the general definition of GeoMaps for different SoS domains, a *GeoMap Metamodel* is developed to conceptualize the geographical information of an SoS, as Figure 2 shows. On the basis of the definitions in Section III, the GeoMap class is defined as an abstract class that aggregates one or more *DimVars* and zero or more *DataVars*. Both variables are defined using the `_SimDataVariable_`, which is an abstract class that represents a generic type for various data. The domains of data possibly assigned to the variable objects can be constrained by concrete domain objects, called `_SimDataDomain_`. The *DimVarDomain* class constrains a *DimVar* object, and the *DataVarDomain* likewise constrains a *DataVar* object. A domain of a dimension variable should be discrete so as to logically locate an object on the map.

B. Developed Processes and Techniques

1) Process for Developing a Concrete GeoMap: With the explained GeoMap metamodel, a concrete (or domain-specific) GeoMap class can be developed by following the process shown in Figure 3. The first step is to identify the dimensions of the GeoMap, such as *xPos* and *floorNum*, which can be defined as discrete dimension variables (*DimVars*) in the second step. As explained in Section III, a dimension variable should specify its domain (*DimVarDomain*) to determine the range of possible values, and a combination of domains of multiple dimensions determines the intended size of the GeoMap. In

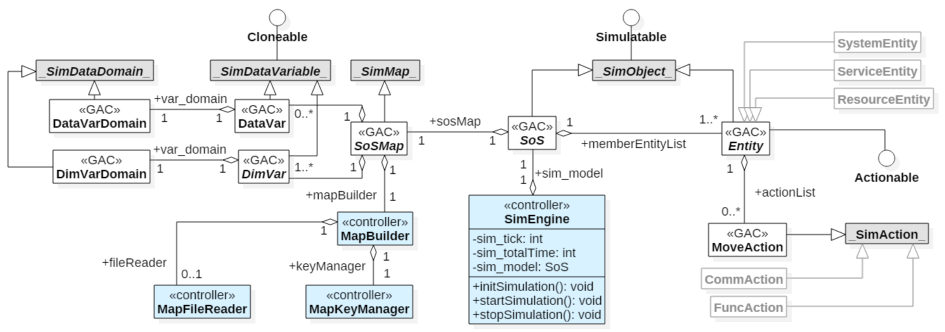

Fig. 4. *GeoMapBuilder-defined Abstract Classes (GACs)* of a Simplified SoS Simulation Model and a Simulation Engine

the third and fourth steps, the graphical properties and data of interest are identified and defined as data variables (*DataVars*) with their domains (*DataVarDomains*). If the variables of the dimensions and data are defined in a map object, HashMap-based location information will be automatically initialized with pairs of keys of all possible location points and default values of the *DataVars*.

2) Initialization and Update of the GeoMap: If an *SoSMap* (a GeoMap of an SoS) object is successfully defined and initialized, then an engineer can either statically or dynamically initialize/update the map. As explained in Section IV-B1, the concrete data of all location points (i.e., GeoLocations) are stored in a hashmap, so search-by-key (*get*) and update-by-key (*put*) operations are supported. However, if some location points need to be updated in batches, the engineers may have to exert considerable effort in the manual initialization or update. For this reason, our model supports the query-based initialization of a GeoMap, as Listing 1 shows. The syntax of the query highly resembles the syntax of SQL-based queries, which use `SET` and `WHERE` keywords. The `SET`-clause (e.g., isWallVar=false) defines queries to assign values to the *DataVars*, and the `WHERE`-clause (e.g., xPosVar==1) specifies the conditions of *DimVars*. The `ALL` keyword can be used or the `WHERE` clause be omitted for an update of all the location points of the map. The assignment should follow the domains of the *DataVars* defined in the map, otherwise, the update will be denied by our map building engine, called *MapBuilder*.

```
// GeoMap : 3*3 tiles
// DimVars: xPosVar (INT:0~2), yPosVar (INT:0~2)
// DataVars: isWallVar (BOOLEAN:TRUE,FALSE)
SET(isWallVar=FALSE)WHERE(ALL);
SET(isWallVar=TRUE)WHERE(xPosVar==1);
SET(isWallVar=TRUE,isChargingStationVar=TRUE)
WHERE(xPosVar==2&&yPosVar==2);
```

Listing 1. Example Queries for Map Initialization

3) Locating Entities/Objects on the Map: A key of the hashmap (*map_locInfo*) is generated as a list of concrete values of *DimVars* to represent a particular GeoLocation. For example, if a map has three *DimVars* (e.g., *xPosVar*, *yPosVar*, and *floorNumVar*) and (3,2) is a location of interest on the third floor of the map, the location can be represented as **`3,2,"FLOOR_3"`**. In our simulation engine, the same key string applies to the location of an object (*obj_location*). Thus, a simulation engineer can initialize the location information of each object using the key string.

C. Implementation: GeoMapBuilder-for-SoS

SoS simulation model. An SoS simulation model based on the *Metamodel for SoSs (M2SoS)* is developed [14]. M2SoS comprehensively conceptualizes containers, entities, and objects that constitute an SoS, and major container classes consist of an SoS, Organization, Infrastructure, Environment, and Map. However, to build a simple model focusing on geographical distribution, we design a simplified version of the simulation model using the minimum set of classes defined in Figure 4. The model introduces major *GeoMapBuilder-defined abstract classes* (GACs), which can be specialized into domain-specific concrete classes constituting a specific SoS.

Among the GACs, classes whose nomenclature is `_ClassName_` (i.e., gray boxes of Figure 4) are abstract classes that cannot be directly inherited by user-defined classes. All the objects and entities defined in an SoS are subclasses of `_SimObject_`, which represent a simulatable object. In the definition of a map, objects, and actions in an SoS, other abstract classes (`_SimMap_`, `_SimDataVariable_`, `_SimDataDomain_`, `_SimAction_`) are also included in the simulation model. Since these simplified GACs are designed to simulate the geographical features of an SoS, geographical classes are defined to follow the definitions in Section III.

According to the definition of GACs, an *SoS* class contains an *SoSMap* object and one or more actionable *Entity* objects. The SoSMap is defined by including lists of *DimVars* and *DataVars*, and the geographical information of the map is managed by a key-based hashmap (*map_locInfo*). From the definition of an SoSMap, every subclass of `_SimObject_` can have its own GeoLocation (*obj_location*). A string-based key representing a location point (e.g., (3,9) for a 2D map) can be used as a particular location of an object. The model largely simplifies the *Entity* class and action classes. M2SoS originally defines different types of entities, such as *Constituent*, *SystemEntity*, *ServiceEntity*, and *ResourceEntity*, but this model abstracts the definition of entities and only considers them as an actionable entity that can perform capable actions. Moreover, among various types of actions (e.g., *FuncAction*

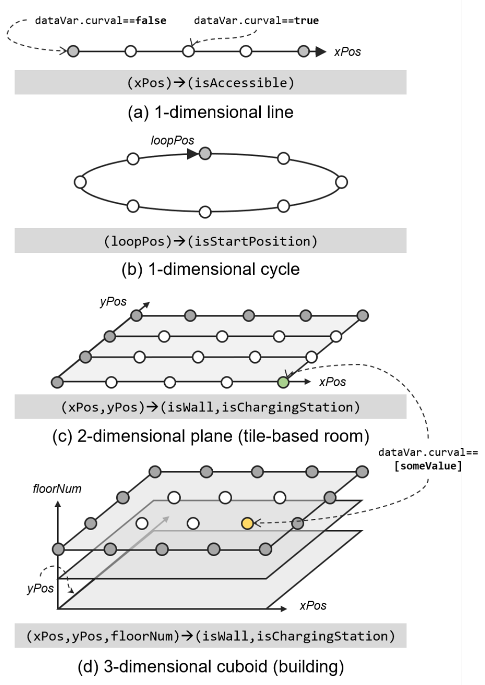
dataVar.curval==false
dataVar.curval==true
xPos
(xPos)→(isAccessible)
(a) 1-dimensional line
loopPos
(loopPos)→(isStartPosition)
(b) 1-dimensional cycle
yPos
xPos
(xPos,yPos)→(isWall,isChargingStation)
(c) 2-dimensional plane (tile-based room)
dataVar.curval==[someValue]
floorNum
yPos
xPos
(xPos,yPos,floorNum)→(isWall,isChargingStation)
(d) 3-dimensional cuboid (building)

Fig. 5. Example Types of GeoMaps for Case Study

and *CommAction*), only the *MoveAction* class is defined as a subclass of `_SimAction_`.

Simulation engine. To validate constituent entities and verify whether a GeoMap is correctly developed in a simulation model, we develop a simulation engine that can perform basic functionalities for the discrete-time simulation. According to our previous work [6], the simulation engine (*SimEngine* in Figure 4) repeatedly runs the simulation model across logical and discrete time (i.e., simulation tick). Currently, the simulation engine mainly focuses on geographical changes and events that occur in a given simulation model. The engine builds a map with the *MapBuilder* controller class, and it can take a *mapInitFile* as an input using *MapFileReader* for the initialization of a GeoMap.

On the basis of the implementation of the GACs and the simulation engine, a simple program and example GeoMaps are developed, and they are currently available on GitHub[1].

V. Case Study

To show and validate the effectiveness of our GeoMap modeling method, we first selected four major types of GeoMaps that can be utilized for the simulation of many SoS cases, as Figure 5 illustrates. Four types are (a) one-dimensional (1D) line, (b) 1D cycle, (c) 2D plane, and (d) 3D cuboid, and the formal definitions of the maps are provided. These GeoMap types appear simple, but an n-dimensional map can be defined by adding any dimension to the maps. The map nodes partially represent the GeoLocations of each map, and the node colors symbolically represent the values of the *DataVars* of each location.

[1]https://github.com/KAIST-SE-Lab/GeoMapBuilder-for-SoS

Among these four types, only the implementation of the last case (d) is introduced as a representative example due to lack of space, but the implementations of all the other cases are included in the GeoMapBuilder project. As Figure 5 illustrates, the cuboid-type GeoMap has three *DimVars* and two *DataVars*, and the variables and the map are defined as specified in Listing 2. For the initialization of the map, Listing 3 states the initialization queries for two *DataVars* for GeoLocations that meet the conditions.

```
defVar xPos(,,[0,4])
defVar yPos(,,[0,3])
defVar floorNum(,,["FLOORB1", "FLOOR1", "FLOOR2"])
defVar isWall(FALSE,,[TRUE,FALSE])
defVar isChargingStation(FALSE,,[TRUE,FALSE])
defMap BuildingMap: (xPos, yPos, floorNum)
                    -> (isWall, isChargingStation)
```

Listing 2. Definition of a Cuboid Map

```
SET(isWall=FALSE, isChargingStation=FALSE);
SET(isWall=TRUE)WHERE(xPos==0);
SET(isChargingStation=TRUE)
WHERE(xPos==3 && yPos==1 && floorNum=="FLOOR2");
```

Listing 3. Initialization Queries for the Cuboid Map of Figure 5-(d)

From the abovementioned definition and initialization process, the cuboid GeoMap can be manipulated or extended easily. For example, if an engineer needs to additionally consider a new dimension (`buildingId`) and a new data (`dustLevel`), the existing *BuildingMap* can be extended to *MultiBuildingMap* by defining a new *DimVar* as showed in Listing 4. Although this is a small GeoMap example, any map can be easily reconfigured either at design-time (statically) or at run-time (dynamically) according to the implementation.

```
defVar buildingId(,,["BLDG_A", "BLDG_B"])
defVar dustLevel(0,,[0,100])
defMap MultiBuildingMap:
       (xPos, yPos, floorNum, buildingId)
       ->(isWall, isChargingStation)
```

Listing 4. Extension of a Cuboid Map

VI. Evaluation and Discussion

Strengths. The definition and metamodel of the GeoMap support scalable and extendable modeling of the geographical information of an SoS. By using two different types of variables (*DimVar* and *DataVar*), our modeling method concisely conceptualizes the overall geographical information of an SoS. The proposed method also enables flexible modification of a GeoMap, as it allows engineers to simply adjust the domains of variables (*DimVarDomain*, *DataVarDomain*). By modifying *DimVarDomains*, an engineer can easily increase or decrease the number of location points of a GeoMap. Similarly, by specifying *DataVarDomains*, one can represent a variety of geographical properties of a location point.

Second, our GeoMap implementation offers an interface for the query-based initialization/update of the map. This can not only support the domain engineering for various SoS domains but also enable the development of predefined maps, templates, and themes. Furthermore, with use of this dynamic/static update of the states/statuses of a map, any simulation engine can easily update the map according to environmental changes, events, or stimulus during a simulation.

Lastly, this study introduces semi-formal definitions of a GeoMap and its properties with a metamodel and formal representations. From them, a model-driven approach can be supported to develop a GeoMap for any type of simulation. If a GeoMap implemented in a simulation model conforms to our GeoMap metamodel, a scalable formalism and corresponding operations can be supported systematically.

Limitations. The current version of *GeoMapBuilder* does not support a map rendering technique and a graphical user interface (GUI). If they are developed, a simulation engineer will be able to easily configure and monitor geographical states for simulation. This limitation can be addressed by extending our open-source tool to include a symbolization method for map rendering and graphical representations. Another limitation can come from the management of map data based on a local memory, which may consume large amounts of computing resources. This issue can be handled with support for database(DB)-based management; in this manner, remote storage is independently utilized specifically for GeoMap management. DB-based management has additional advantages in the use of set and Boolean algebras and nested/complicated query processing. Finally, an SoS ecosystem of an SoS can have geographical constraints, which need to be properly included in a GeoMap model. To prevent the building of an infeasible map, other controllers (e.g., *ConstraintBuilder* and *Rule(Conflict)Checker*) can be implemented in a domain-specific way, on the basis of a specific scheme. This problem can be addressed via interweaving between a GeoMap model and real data models, which will enable DataVar objects to be more realistically designed.

Future Work. Due to these limitations, future studies and investigations are needed. First, a mapping technique between a physical SoS map/world and a logical GeoMap should be developed. The mapping may require specific data models, domain knowledge, and thorough analysis of the application domain. While developing a logical GeoMap, SoS engineers should consider the following characteristics of physical SoS maps: 1) *continuity* (temporal and spatial), which is a fundamental characteristic of the physical world; 2) *uncertainty*, which is inherent in data variables and real-world events; and 3) *interdependency*, which refers to mutual influences between different geographical locations. These characteristics should be rigorously analyzed and engineered using a proper abstraction methods and data analyses.

VII. Conclusion

In order to perform a more realistic modeling and simulation (M&S) of a System-of-Systems (SoS), it is necessary to systematically represent, design, and model geographical distribution of constituents. Most of all, designing a geographical map must take precedence in order for the simulation of behaviors and interactions of the distributed SoS entities. This study, for this purpose, focuses on a general-purpose modeling method to represent geographical information of an SoS, including a geographical map (GeoMap), object locations, and geographical changes and events. To support systematic GeoMap modeling, we proposed a GeoMap meta-model and developed *GeoMapBuilder-for-SoS* that supports scalable definition and management of a GeoMap. Through case studies, this study shows the applicability, effectiveness, and scalability of our modeling method, and effectiveness and scalability are also demonstrated. Future work is needed to support additional operations a GeoMap class (e.g., union, distance, correlation, etc.) and automated GeoMap code generation.

Acknowledgement

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2021-2020-0-01795) and (No. 2015-0-00250, (SW Star Lab) Software R&D for Model-based Analysis and Verification of Higher-order Large Complex System) supervised by the IITP(Institute of Information Communications Technology Planning Evaluation).

References

[1] M. Jamshidi, System of systems engineering: innovations for the twenty-first century. Wiley, 2008, vol. 58.
[2] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: Basic concepts, model-based techniques, and research directions," ACM Comput. Surv., vol. 48, no. 2, Sep. 2015. [Online]. Available: https://doi.org/10.1145/2794381
[3] M. W. Maier, "Architecting principles for systems-of-systems," Systems Engineering, vol. 1, no. 4, pp. 267–284, 1998.
[4] A. Babu, S. Iacob, P. Lollini, and M. Mori, "Amadeos framework and supporting tools," in Cyber-Physical Systems of Systems. Springer, 2016, pp. 128–164.
[5] I. Sommerville, D. Cliff, R. Calinescu, J. Keen, T. Kelly, M. Kwiatkowska, J. Mcdermid, and R. Paige, "Large-scale complex it systems," Communications of the ACM, vol. 55, no. 7, pp. 71–77, 2012.
[6] S. Park, Y. Shin, S. Hyun, and D. Bae, "Simva-sos: Simulation-based verification and analysis for system-of-systems," in 2020 IEEE 15th International Conference of System of Systems Engineering (SoSE), 2020, pp. 575–580.
[7] C. Portele, "Ogc geography markup language (gml)–extended schemas and encoding rules," in Open Geospatial Consortium Inc. Citeseer, 2012.
[8] "Keyhole markup language," Aug 2020. [Online]. Available: https: //developers.google.com/kml/documentation/kmlreference
[9] G. Gröger, T. H. Kolbe, C. Nagel, and K.-H. Häfele, "Ogc city geography markup language (citygml) encoding standard," 2012.
[10] J. L. Filho and C. Iochpe, "Specifying analysis patterns for geographic databases on the basis of a conceptual framework," in Proceedings of the 7th ACM international symposium on Advances in geographic information systems, 1999, pp. 7–13.
[11] G. B. Sampaio, F. R. Nalon, and J. Lisboa Filho, "Geoprofile-uml profile for conceptual modeling of geographic databases." in ICEIS (3), 2010, pp. 409–412.
[12] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in The 21st IEEE International Conference on Intelligent Transportation Systems. IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/
[13] "Anylogic." [Online]. Available: https://www.anylogic.com/
[14] Y.-M. Baek, J. Song, Y.-J. Shin, S. Park, and D.-H. Bae, "A meta-model for representing system-of-systems ontologies," in 2018 IEEE/ACM 6th International Workshop on Software Engineering for Systems-of-Systems (SESoS). IEEE, 2018, pp. 1–7.