A DSL-Driven Development Framework for Components to Provide Environmental Data in Simulation based Testing

Liqun Wu and Axel Hahn

University of Oldenburg, Ammerländer Heerstraße 114-118, 26129 Oldenburg, Germany

Keywords: Simulated Environment Description Language, CIM-PIM Transformation, Model-Driven Development.

Abstract: Developing components that produce data representing simulated environments for spatial-aware simulations could be difficult and error-prone. Knowledge of the required outputs of these components and computational models of the environmental phenomena are often held by different roles in the development. Miscommunications may appear among involved roles due to their different perspectives to view environmental phenomena. Consequently, requirements of simulated environments in simulation scenarios may not be correctly preserved in the developed components. This paper presents a domain-specific development framework to overcome this problem. It focuses on bridging the gap between human-view requirement descriptions of simulated environments and system-view component design models to produce digital representations of these environments. It specifies a CIM (Computation-Independent Model) -layer language which supports system of interest modelers to document required context of simulated environments in their simulation scenarios in a half-formal manner. Transformation rules from these CIMs are established to derive necessary data structures and computation flows as PIM (Platform-Independent Model) -layer models of simulated environment components. These transformations are further combined with general Model-Driven Development (MDD) solutions to create platform-specific component skeletons.

1 INTRODUCTION

In a computer simulation, the system of interest is often modelled with the logic about how the system behaves in reaction to the influence from phenomena in its situated environment. At an execution step of this simulation, a component should feed the system model with data of these phenomena, which are needed by the system model to compute a new state. This component provides the simulated environment (Klugl, Fehler, & Herrler, 2004) in the simulation.

The environment component may need to enclose realistic observation data or its own simulation models of the involved phenomena types. Due to the inherited complexity from their real-world counterparts, these data and models are not straightforward to be adjusted and integrated into the simulation by non-experts. Safety-critical simulations in the marine domain provide a typical example. Maritime systems situate in the environment with spatio-temporal varied phenomena such as other artificial systems, wind and ocean currents. The influences of these phenomena cannot be ignored in safety-critical simulations, while modelling such complex phenomena are likely beyond the expertise of the system of modelers.

Thus, environment components in these simulations may have to be developed by experts other than the system modelers. However, for acquiring meaningful simulation outcomes, environmental data produced by such a component should match simulation scenarios that the system modelers want to execute. Thus, requirements about the simulated environments in the scenarios must be communicated between system modelers and experts who are able to develop this environment component. Mismatches may appear during the communication due to different perspective of involved roles to view the environmental phenomena. This could cause that the developed component does not correctly preserve requirements from system modelers. The produced data may miss some aspects that are required inputs of the system model. Further, the data values produced in an execution may not match the expected environmental conditions of the executed scenario.

To overcome the above-mentioned problems, this paper proposes a language-driven framework to assist the development of environment components in
simulations. It focuses on bridging the gap between conceptual level requirements of simulated environments and design models of environment components, which is comparable to the view switch when transforming domain-oriented CIMs to system-oriented PIMs in Model-Driven Architecture (MDA) (OMG, 2014). This framework follows the MDA principles. It uses Domain-Specific Languages (DSLs) (van Deursen, Klint, & Visser, 2000) to describe simulated environments as CIMs and environment component models as PIMs. Transformation rules are established between these two types of models, so that necessary elements of an environment component to produce a described environment can be derived from the description of the environment via transformations.

The rest of the paper is organized as follows. Section 2 reviews existing works related to the presented framework. Section 3 gives an overview of this framework. Section 4 and 5 introduce modelling languages used in the framework and transformations among them. Section 6 shows a prototypical implementation of the framework, followed by a use case to demonstrate the transformation process. Section 7 summarizes the contributions of this framework and discusses open issues.

2 RELATED WORK

Most researches on CIM-PIM transformations focus on the business processes and requirements modelling. These works rely on OMG-managed modelling languages to describe CIMs and PIMs, as well as transformation languages to specify transformations among them. At the CIM layer, the business processes are often expressed by BPMN (OMG, 2011) or its specialization (Hahn, Panfilenko, & Fischer, 2010; Rodriguez et al., 2011; Bousetta, El Beggar, & Gadi, 2013; Kriouile, Addamssiri, & Gadi, 2015; Rhazali, Hadi, & Mouloudi, 2015a). Some approaches also model functional requirements as use case diagrams. (Gutiérrez et al., 2008; Bousetta et al., 2013; Kriouile et al., 2015) PIMs in these researches usually cover the structural aspect and the behaviour aspect of a system. The former is frequently expressed by class diagrams (Bousetta et al., 2013; Kriouile et al., 2014; Rhazali et al., 2015a), while activity diagrams (Gutiérrez et al., 2008), sequence diagrams (Bousetta et al., 2013) and state machines (Rhazali, Hadi, & Mouloudi, 2015b) have been used to express the latter. Some approaches also define specialized metamodels for describing PIMs, such as the PIM4Agent from (Hahn et al., 2010) and DSLs used by (De Castro, Marcos, & Vara, 2011).

Transformations in existing researches are formulated using QVT languages (Gutiérrez et al., 2008; Rodriguez et al., 2010; Kriouile et al., 2014; Kriouile et al., 2014) or ATL (Hahn et al., 2010; De Castro et al., 2011; Rhazali, Hadi, & Mouloudi, 2016). Most of them have multiple steps. The overall transformation chains of these researches are usually performed in a semi-automatic manner with human interference between steps to improve the model quality.

Similar researches can be found in other application domains, e.g., for the development of data warehouses (Mazón, Pardillo, & Trujillo, 2007) and Web applications (Kraus, Knapp, & Koch, 2007; Fatolahi, Somé, & Lethbridge, 2008). Existing researches provide the foundation on the components which our framework should have and references about how it may be built. However, they focus on other context domains.

In the context domain we are interested in, human-oriented DSLs to express spatial phenomena based on common sense conceptualizations have been proposed, such as database languages using the “moving objects” concept (Güting & Schneider, 2005), the general type “field” to represent and operate on spatio-temporal data (Camara et al., 2014) and core concepts for spatial computations (Kuhn & Ballatore, 2015) etc. Their implementations are often embedded in spatial DBMS, GIS software or code libraries. They aim to raise the usability of spatial database and systems by hiding a fixed implementation behind cognitive level concepts. On the other side, various system-oriented models have been adapted as ISO/TC 211 (ISO, n.d.) and OGC (Open Geospatial Consortium, n.d.) standards to support to build and exchange spatial data or information services. Models of these standards focus on spatial data sharing and system interoperability. These researches provide fundamental terms of spatial representations that we can utilize. These models are defined at various abstraction levels and are not clearly aligned with the software development phases. The development of environment components in simulations with or without using some of these models is still solved in a case-by-case manner.

3 THE PROPOSED FRAMEWORK

Functional scenarios described at the conceptual level
are determined in the requirement analysis phase when developing computer simulations. (Menzel et al., 2018). Our research classifies the context of description about environmental phenomena in the functional scenarios based on conceptual forms of phenomena and types of their exhibited changes. After that, necessary elements in the simulation program for producing the described context type are identified. The proposed framework in this paper is built on the research outcome and MDA as introduced below.

First, a DSL Simulated Environment Description Language (SEDL) is specified to describe the required context of simulated environments as CIMs. A SEDL description corresponds to a simulation program. It is half-formal with application-specific requirements captured by free text and enclosed by a SEDL description item. A phenomenon expressed in SEDL corresponds to all possible phenomena instances of the same type that can be produced by a component under development.

Then, mapping rules are established from SEDL descriptions to PIM-layer component models in UML. They enable automatic transformations which derive artefacts of environment components from SEDL descriptions. Three types of sub models are included in the PIMs: the configuration schema that describes the parameters to be set for running a component; the data structure that carries state values of computed phenomena during simulations; the computation flows that compute the states of computed phenomena during simulations; the component under development.

Further transformations from PIMs to models and code in a specific platform can utilize general MDA solutions, since they are rather technical refinement which does not involve the view switch or the domain knowledge. An implementation of the framework supports to describe simulated environments as SEDL descriptions, execute transformations of SEDL descriptions to derive component models and code skeletons. Application-specific requirements captured in the free text are preserved within model elements or code units to guide implementation.

4 LANGUAGES FOR SIMULATED ENVIRONMENTS

This section introduces modelling languages used in this framework at the CIM layer and the PIM layer.

4.1 Simulated Environment Description Language (SEDL)

Terms in the CIM-layer language SEDL classifies pieces of descriptions about simulated environments in simulation scenarios. All SEDL terms are made as subtypes of DescriptionItem which has an attribute description to capture application-specific requirements in the free text. Besides, each type has attributes to capture the type-specific context.

Figure 1: The Description Structure of SEDL.

SEDL uses an entity-property hierarchy to organize description pieces as show in Figure 1. An instance of the entry term SimulatedEnvironment encloses all description pieces about an environment component. Terms that describe environmental phenomena are subtypes of Configurable. Their instances may have ConfigurableParameter-s, each of which describe a modifiable condition about this Configurable for different executions. Two concrete types can be chosen to describe a type of environmental phenomena. SpatialIndividuality describes a type of phenomena which appear as an identifiable individual substance in space. FieldOfIndividualities describes a set of individuality from the same type with no significant member, such as a group of randomly moving ships. All members in this field together exhibit some spatial pattern. Which subtype to choose depends on the simulation scale, the way that system modelers view the phenomenon etc. It does not mean to reflect the “truth” of the real-world entities.

SEDL supports to classify description of changes that environmental phenomena should exhibit in simulation scenarios.

The CharacteristicVariation is used to describe the value variation of a characteristic index or several correlated indexes among a whole set of instances of a phenomenon type, e.g., some initial state. The described variation is a value distribution of these indexes. A CharacteristicVariation instance belongs to an EnvironmentalPhenomenon E. If E is
the member of a **FieldOfIndividualities** F, when creating an instance of the phenomenon described by F, relevant index values of its members should be drawn from the distribution. Otherwise, when creating an instance of the type described by E, relevant index values of this instance should be drawn from the distribution.

Change types about an individuality that shall be perceived by human are identified as shown in Figure 2. An instance of them expresses a change pattern which a phenomenon type should exhibit during simulations. Essentially, it is the pattern of difference about one characteristic of a phenomenon P, when another characteristic of P is altered in a controlled way. It can be viewed as a relationship between states of this phenomenon in two value domains: \( A \rightarrow B \).

Change types are identified based on rational combinations of the A’s type and B’s type. The types of the value domains can be spatial locations, temporal locations and thematic properties (Guttag & Horning, 1978). Further, location differences in high-level descriptions shall be approximately viewed as self-referenced and externally-referenced. E.g., the “Movement” in Figure 3 is the observed spatial location differences relative to an external reference (often the earth) over time. The application-specific pattern of a change description instance is informally captured by its description slot.

### 4.2 PIM-Layer Metamodels of Environment Components

In this framework, UML is used as the metamodel to describe PIMs. To facilitate the object-oriented code generation, PIMs are expressed in the class view. A computation flow is presented as a uml:Class and a uml:Operation in this class. Each of its computation units is presented in the class as a uml:Operation and statements within the body of the flow operation that invokes the unit operation.

The framework specifies a UML profile to provide a set of stereotypes for describing structural models of environment component more concisely as shown in Figure 3. It includes subtypes of **SimulatedFeatureType** which represent the data structure to hold state values about a phenomenon type during simulation executions. Each subtype restricts following aspects of a class: 1) how the geometry of a phenomenon type is represented by this class; 2) how property values representing the phenomenon’s thematic characteristics are linked to its geometry. These types are introduced in a two-dimensional context here. Nevertheless, each of them has its three-dimensional counterpart. They are specified based on popular spatial computation paradigms. For instance, subtypes of

![Figure 3: Stereotypes of Simulated Feature Types.](image-url)
CollectiveFeatureType are used to represent phenomenon types that are computed as a set of units, each of which occupies a spatial location. Such representations are used in multi-agent spatial simulations. Each subtype of the CollectiveFeatureType regulates a spatial representation of the units, including the type of their geometries and spatial relationships among them.

Besides, SpatialFunction is a class which holds the function to represent the form of a spatially heterogeneous theme. It shall appear as the attribute type of a GlobeFeature or a LocalFeature.

5 CIM-PIM TRANSFORMATIONS

This framework specifies three CIM-PIM transformations to derive design models of environment components from SEDL descriptions.

First, the transformation SEDL2Config generates a configuration schema whose structure is aligned with the structure of the input SEDL description. In general, it transforms each Configurable to a uml:Class. The parameters of the Configurable are mapped to attributes of the output class.

Then, the transformation SEDL2Structure derives data structure models of environment components based on following principles:
1) A class stereotyped with a SimulatedFeatureType subtype is created for each EnvironmentalPhenomenon. The applied stereotype is derived from the type and the spatial form of EnvironmentalPhenomenon and its change types. For instance, a FieldOfIndividualities whose members are 0-dimensional entities and have changes involving locations is transformed to a PointSetFeature.

2) For each ThematicProperty of a SpatialIndividuality, an attribute is added to the output feature class, or the output unit class when this Spatial Individualities is the member of a FieldOfIndividualities.

3) A SpatialFunction class is created for a ThematicValueCollection to hold the distribution function that determines a value at a spatial location. This class is set to be the attribute type created from the ThematicProperty with this distribution.

The transformation SEDL2Computation uses the output of the above two transformations and the input SEDL description to generate computation units and build computation flows with these units, which update states of simulated feature properties.

First, it creates a uml:Class for each EnvironmentalPhenomenon Ep. An uml:Operation is added to the class for each of its CharacteristicVariations and individuality changes.

Then, a directed graph is derived based on individual changes of Ep, or of its member if Ep is a FieldOfIndividualities. Nodes of the graph represent properties of Ep. Edges correspond to the relation between the connected nodes expressed by a change description, e.g., an edge \( t \rightarrow tp \) denotes a ThemeDynamics of its ThematicProperty \( tp \). A reference to the generated operation of this ThemeDynamics is stored with the edge. This graph implies the appropriate order to update properties of an instance phenomenon. Cycles in the graph are detected and replaced by a compound node. The compound node corresponds to a sub-computation that determines the values of the nodes in the cycle.

An uml:Operation is generated, which executes the computation flow to update states of a simulated feature at a simulation step. It contains statements that invoke the operations to update the attributes of the feature data object held by the computation class, in a sequence based on the topological order of the generated graph. This flow applies to an individual feature. For a FieldOfIndividualities, the generated operation updates a unit of the CollectiveFeatureType generated from it. An iteration is also generated to execute this function over units of this CollectiveFeatureType.

It is suggested to have a manual refinement on the intermediate output to bring in more details which is not derivable. For instance, the unit geometry of a TessellatedFeature is decided by the developers. The transformation generates a TessellatedFeature with denotation that the applied stereotype should be replaced by a subtype of the TessellatedFeature.

6 USE CASE

A prototype of the framework is implemented based on Eclipse Modeling Framework (EMF) (Steinberg et al., 2008) as summarized in Figure 4.

The SEDL Abstract Syntax is encoded as an Ecore (Steinberg et al., 2008) model. The SEDL2Config and SEDL2Structure are implemented as ATL transformations. The data structure profile is encoded using the UML2 plugin (Eclipse, n.d.). The Platform-Specific Translator can be varied in different implementations of the framework. In this prototype, they are three code generators which create Java files for developing Eclipse plugins. The EMF code generator is used to create code for configuration schemas. The other two generators are implemented as Acceleo (OEBO, n.d.) templates. PIMs from
SEDL2Computation are made implicit in the prototype. The generator for computation models uses SEDL descriptions as inputs and produced Java files. Operations on graphs are implemented by Java. The simulation library Mason (Luke et al., 2005) and its spatial extension GeoMason (Sullivan, Coletti, & Luke, 2010) are used in the output code to deal with simulation routines and spatial-related issues.

The SEDL Toolset is implemented using EMFText (Heidenreich, Johannes, Karol, Seifert, & Wende, 2009). A textual SEDL Concrete Syntax is specified in a file with “.cs” extension and referenced to the SEDL Ecore model. EMFText creates various language facilitates from this “.cs” file, which are customized afterwards. The prototype can be used to write SEDL descriptions in the concrete syntax and store them in files with the “.sedl” extension. The transformations are integrated into the SEDL processor, which can be invoked to execute a file with this extension.

A demonstrative use case is presented, which uses the prototype to develop an environment component for a vessel simulation. This component is implemented as an Eclipse plugin. The functional scenario is that a cargo vessel executes a planned path under various weather conditions. During the simulation, the environment component should simulate environmental data required by the vessel model execution with desired conditions, e.g., force of wave, information of other ships etc.

The required environment is documented in the “Sea.sedl” file using the implemented textual syntax. Various artefacts are generated from this file by the framework prototype. Figure 5 shows the transformation flows. Classes in the “seaCon” package are generated by EMF.

A DSL-Driven Development Framework for Components to Provide Environmental Data in Simulation based Testing

Field of Individuality BackgroundTraffic {
  "A set of ships move in the environment. The number of ships should be set by users to test the cargo ship's behaviours in various traffic density."
  parameter: numberOfShips, Number "the number of ships"
  Spatial Individuality Ship {
    type: Point
    Characteristic Variation: Random Speed {
      "The speed of a vessel should be randomly picked from a set of possible values that are specific to different ship types. The probability that a value is picked should equal to the ratio of the corresponding ship type within whole traffic flow at the area of interest."
      index: speed
      Partially omitted due to space limitation
    }
  }
  Thematic Property Moving Direction {
    Dynamics! Correlated Value {
      "The moving direction of a ship should turn an angle relative to the direction of the last step. The angle should be drawn from a normal distribution."
      parameter: mean, Number "mean of the turning angle, 0-50 degree"
      variance, Number "variance of the turning angle, 0-50 degree"
    }
  }
}

Figure 6 shows a description piece of a phenomenon type in the “Sea.sedl” file for illustration, in which the free-text descriptions are displayed in green. The key word “Dynamics” in the concrete syntax corresponds to the ThemeDynamics.

Figure 7 shows a visualized diagram of the generated configuration schema for the “BackgroundTraffic” in Figure 6 (left) and the data class for a ship in the background traffic (right). Attribute access methods are omitted due to space limitation. The PIM layer data class for the “BackgroundTraffic” is mapped to a PointSetFeature as Section 5 introduced. In the prototype, it is further transformed to a GeomVectorField object whose geometry is initialized as a set of points managed by the generated class “ComputeSea.java” in Figure 5.
A domain-specific framework is presented in this paper, which eases the development of environment components in computer simulations by following means. First, it provides a CIM-layer DSL SEDL as a communication tool to discuss and document the requirements of components under development. Second, it establishes mappings between conceptual contexts of simulated environments expressed in SEDL and necessary artefacts in computer programs in order to produce them. Thus, system-perspective design models can be derived from human-perspective requirements. Third, it enables automatic generations of program models from the requirements descriptions. Architectural code can be further generated with only application-specific methods to be filled. The application-specific requirements are preserved within corresponding code units to guide the implementation.

Our future work focuses on the optimization of transformations to the chosen technical platforms, the test of the implemented framework with our marine simulation development and the strategies to reuse programs of environmental phenomena developed within this framework in other simulations.

**REFERENCES**


A DSL-Driven Development Framework for Components to Provide Environmental Data in Simulation based Testing


