Simulation Independent Model Configuration

F.P.J. van Wermeskerken, G.R. Ferdinandus, T.W. van den Berg, K. van den Bosch, R.M. Smelik, H. Henderson
Netherlands Organisation for Applied Scientific Research (TNO)
The Netherlands
{freek.vanwermeskerken, gwen.ferdinandus, tom.vandenberg, karel.vandenbosch, ruben.smelik, henk.henderson}@tno.nl

Keywords: simulation scenario, model configuration, ontology, OWL

ABSTRACT: *The models of all entities involved in a simulation need to be configured to behave in line with the purpose of the scenario. Simulation model configuration is quite complex for several reasons. For one, a simulation model such as a human behavior model or a sensor model has many detailed parameters and specialist knowledge is needed to configure and use these models appropriately. The people that develop scenarios for simulation systems are seldom specialists in the underlying simulation models. Moreover, different simulation systems that model the same or similar concept like a human or a specific type of vehicle, in practice all use different models and model configuration parameters. Configuration parameters for similar concepts have different names or meanings in different systems and simulation model configurations cannot be exchanged between simulation systems.*

This paper investigates the concept and envisioned use of a so called "Simulator Independent Model Configuration" (SIMC) language based on the ontology language OWL to describe simulation model configurations in a simulation system independent way. Using the SIMC language it should, for example, be possible to model system-neutral simulation model configurations that can be translated into system-specific simulation model configurations in a systematic fashion. Furthermore, the semantic properties of OWL should open possibilities for reasoning about concepts, properties and parameters, and should provide the capability to infer further information. This paper presents several use-cases and an analysis of a case study to evaluate the suitability of the OWL language for defining simulation models configurations.

1 Introduction

1.1 Background and problem description

In order to execute a simulation, simulation systems need to be initialized with what is called an executable simulation scenario [1]. Such a simulation scenario describes the initial set of conditions and may also include pre-planned courses of action and significant events that should happen during the simulation execution. The initial conditions include for example information on the environment (such as weather conditions), and information on the status and position of units and equipment within the environment. This information is used to initialize the simulation models in the simulation system.

Besides the information specific to the simulation scenario there are often many other parameters for simulation models in (military) simulation systems. Examples include: detailed information on units and equipment, damage assessment information for munition detonations, various parameters for the characteristics of sensor or weapon systems models, and all kinds of parameters to describe entity behavior. These model specific parameters determine to a large extent the resulting behavior of simulation entities, their interactions, and the effects of their interactions at run-time, and thus ultimately the outcome and validity of a simulation execution. Model specific parameters are therefore important for the description of a simulation scenario and the ability to reproduce outcomes in different simulation executions.

Unfortunately, simulation models and their corresponding model configurations are mostly simulation system specific and model specific parameters (assuming they are not hardcoded) are configured using dedicated user interfaces or non-standardized configuration files. Some parameters may be hardcoded and not accessible to the regular user at all. Moreover, different simulation systems that model the same or similar concepts such as a "human", a "sensor", or a "tank" in practice all have their own unique simulation models for these concepts and all use different parameters (with different names and semantics) to configure these. While standards exist to exchange simulation scenarios between simulation systems (i.e. Military Scenario Definition Language (MSDL)), there are currently no standards to describe model specific parameters, exchange these between simulation systems, and reference these from simulation scenarios.

As a result personnel that has to configure models for a specific simulation system have to become specialists regarding the particular system. They have to invest significant study and experimentation time in order to be able to use and configure the models to suit the needs of the scenario.

Ideally, simulation models should all use the same source for model parameters in order to have a shared understanding what these parameters are, their semantics, but also to mitigate differences in behavior, interactions and effects later at run-time. A comparable issue exists for environmental data, where – ideally – simulation systems in a simulation environment should identify and use a single baseline of environment source data for their simulation models, or increase the risk on correlation errors between different simulation systems at run-time.

1.2 Objective of this paper

This paper investigates the use of the web ontology language OWL as the base for a simulation system independent language for the description of simulation model parameters and their semantics. These descriptions can serve as the reference source of information for simulation models and their model specific parameters. In this paper we call this language the *Simulator Independent Model Configuration* (SIMC) language. Furthermore, the semantic properties of OWL should provide possibilities for reasoning about concepts, properties and parameters, and the capability to infer further information.

Using the SIMC language it should therefore be possible to describe a simulation model configuration in a system independent way and translate such a description into model specific parameters for a certain simulation system. An example of model specific parameters is a *Simulation Model Set* (SMS) file for VR-Forces, a Computer Generated Forces simulation system from VT MAK [2]. The collection of model specific parameters for a certain simulation system is in this paper called a *Simulator Model Configuration* (SMC). The relationships between executable simulation scenario (i.e. MSDL), simulator independent model configuration, simulation system and executable (simulation) models are illustrated in Figure 1.

This paper investigates and evaluates the usefulness of OWL for SIMC with the help of several use cases, as will be described in later sections.

1.3 Related work

Some of the work for this paper was inspired by the SISO ANDEM Study Group [3]. One of the ideas of ANDEM was to represent an architecture specific Simulation Data Exchange Model (SDEM, such as an HLA FOM file) with a common representation scheme, resulting in a so called Architecture Neutral Data Exchange Model (ANDEM). An architecture neutral SDEM could then be translated to an architecture specific SDEM for different simulation architectures such as DIS or HLA. By using OWL as language for ANDEM it should also be possible to reason about architecture neutral SDEMs and compare them. Both the SDEM and the architecture neutral SDEM are focused on the data exchange between simulation systems at runtime (see definition of SDEM in [4]). This paper focuses however on the description of simulation model configurations and the configuration of executable (simulation) models pre-runtime.

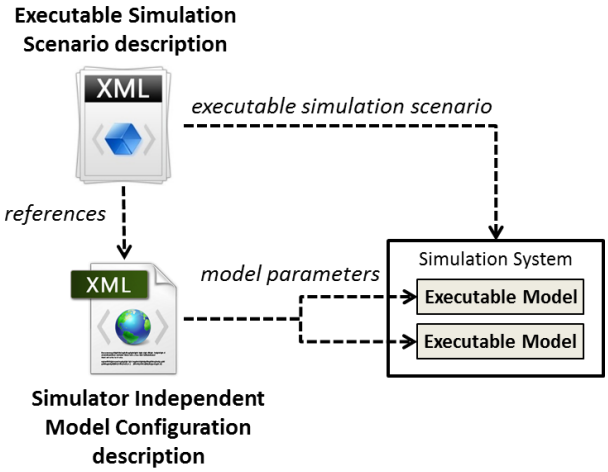

Figure 1: Scenario and model configuration.

1.4 Document overview

The remainder of this paper is structured as follows:

Section 2: provides the objectives and use cases of the SIMC language. One of these use cases is explored in more detail in the case study described in Section 4.

Section 3: provides a brief introduction to OWL and describes how OWL is used as base for SIMC.

Section 4: describes the case study that explores one of the use cases from Section 2 in more detail. A simple example is used to illustrate SIMC and to discover the advantages, disadvantages, and issues of using OWL as Simulator Independent Model Configuration (SIMC) language.

Section 5: discusses the findings from the case study.

And finally Section 6: provides a summary and conclusions, and further recommendations.

2 Objectives and Use Cases

2.1 Objectives of SIMC

The main objectives of the SIMC language are:

- Provide a common language (called SIMC language) to specify, compare and reuse Simulator Model Configurations (SMCs) of different simulation systems.
- Describe a SMC of a simulation system in a system independent way.
- Provide support for the translation of a SMC of one simulation system to a SMC of another simulation system.
- Describe the characteristics and capabilities of simulation models in a semantically-precise way.
- Capture knowledge about simulation models and provide the possibility to reason about simulation models and infer new knowledge.

To address the problems posed in the previous section a system is required, in addition to the SIMC language, that enables users to create, edit and share Simulator Independent Model Configurations (SIMCs). Moreover the system should also be able to reason about SIMCs and facilitate the systematic translation from a SIMC to a SMC.

The following section describes the concept of this system and how we envision its use.

2.2 SIMC System and use cases

The *SIMC System* is a service-based system that allows users to create, edit and download SIMCs. Figure 2 provides a high-level overview of the functional components of the system concept and the actors. The system consists of three main functional components:

- The **SIMC repository** in which all SIMCs are stored.
- The **Simulation system model repository** containing descriptions of the SMC-structure of all models of the specific simulation system. These structures can be used by the SIMC system to support the translation of a SIMC into a SMC.
- The **SIMC reasoner application** that supports the translations between a SIMC and a SMC and between SMCs of different simulation systems.

The following actors are identified:

Scenario builder: develops executable simulation scenarios using a scenario development tool. This scenario development tool is integrated in the simulation system including support for interacting with the SIMC System. The Scenario builder uses the SIMC repository to define and export SMCs that can be used in his specific simulation system. The Scenario builder requires knowledge of the simulation system, the SIMC System, and the conceptual scenario he needs to implement, including an overview of all relevant entities and their desired behavior [1].

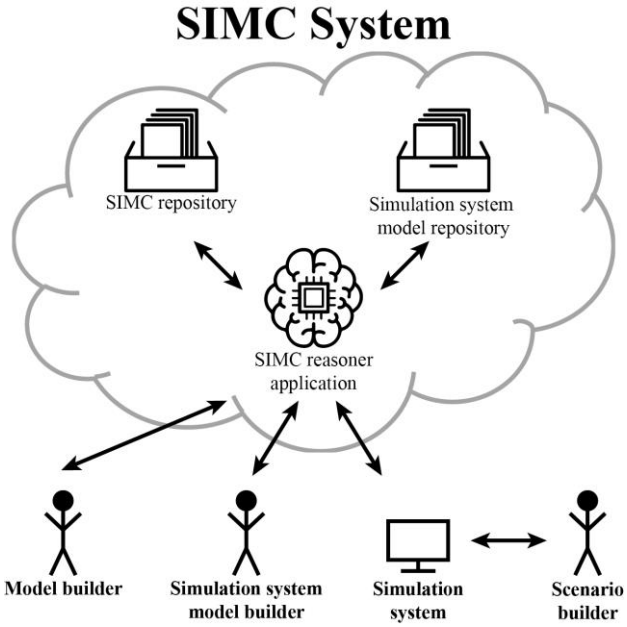

Figure 2: overview of functional components of the SIMC system and the relevant actors.

Model builder: designs and implements entity models including their configuration for use in a simulation system. The Model builder designs the models that the Scenario builder can use in a simulation scenario. The Model builder uses the SIMC language to describe and share model configurations in a system independent way. He may use other SIMCs as inspiration for configuring his own models. The Model builder requires knowledge of the SIMC language and the properties of the entities he is modeling.

Simulation system model builder: uses the SIMC language to develop general SMC structures for the models used by a specific simulation system. These are linked to the same concepts and definitions used in the SIMCs developed by the Model builders. The SIMC System uses the SMC-structures to automatically convert SIMCs to SMCs for the models of that simulation system. The Simulation system model builder requires knowledge of the SIMC language and the specific simulation system.

The following subsections describe several use cases to illustrate the interaction between the actors and the SIMC System.

2.2.1 Scenario builder

For the Scenario builder two use cases are described, one where he creates a new SMC based on a SIMC and one where he benefits from the ability to exchange model configurations between systems.

Use case 1: Create new SMC

Context
Peter is a Scenario builder for a specific simulation system. He has to build a scenario containing a specific type of airplane (MiG-29) but his simulation system does not yet have the required model configuration. However, his simulation system does have a generic fighter plane model. If Peter has detailed information regarding the MiG-29 he could create a model configuration for it by copying the generic fighter plane model configurations and changing the appropriate parameters. Since Peter is not an expert regarding the MiG-29 he uses the SIMC System to get the information he needs while saving time and effort and ensuring the quality of the model.

Interaction
Peter uses the search tools from the SIMC System to find a SIMC of a related entity that he might use to define the model configuration for the MiG-29. Peters selects a SIMC, this SIMC might contain references to SIMCs of other entities for various options regarding sub-components relevant to the MiG-29, like sensor and weapon systems. Peter selects the components and the level of detail that are relevant for his scenario and then exports the created MiG-29-SIMC to a SMC for his specific simulation system. The SIMC System is able to automatically translate (part of) the SIMC to the required SMC. If the SIMC System encounters a parameter that cannot automatically be translated, it prompts Peter for inputting the mapping by hand.

Use case 2: Reference SIMC instance from simulation scenario

Context
Two Scenario builders, Peter and Jim, each working with their own simulation system, want to exchange part of a scenario. Peter saves the scenario using a new version of the simulator independent MSDL standard that supports SIMC annotations. In these annotations references can be stored to the SIMCs describing the model configurations of the simulation entities. The SIMC system can utilize these references to translate the scenario specific model parameters from one simulation system to the other.

Interaction
Peter's simulation system has fully integrated support for SIMC. When he adds an entity to the scenario a reference is stored to the original SIMC on which the configuration for this entity is based. If Peter changes an entity's default parameter to fit the scenario, e.g. sets the fuel level to 50%, the new parameter value is stored separately. When Peter exports the scenario into (the SIMC enabled) MSDL format the SIMC references and scenario specific parameter values are exported as well. When Jim imports the scenario in his simulation system he is prompted to have the system automatically convert the referenced SIMC into a SMC specific for his system. The SIMC System automatically translates the SIMC, ensuring that the scenario-specific parameter values such as fuel level are implemented in the new SMC.

2.2.2 Model builder

This subsection describes a single use case for a Model builder.

Use case 3: Create new SIMC

Context
Catherine is a Model builder. She has created a model of a new entity type for a specific simulation system. She decides to store the model configuration using SIMC. By creating a new SIMC rather than a SMC the model configuration can be shared with other users who can benefit from the knowledge and also provide valuable feedback.

Interaction
Catherine logs into the SIMC System. Using the online editor she opens a SIMC from the repository that describes a concept similar to the entity that she wants to model. She uses it as a base for her SIMC. First Catherine selects the parts of the configuration that she wants to reuse and then she extends the configuration with additional parameters relevant to her purpose for the model. Some parameters can be defined by referring to concepts defined in other SIMCs describing related entities. Catherine's model may also contain concepts not yet present in the SIMC System. She adds these concepts to the SIMC System, including a clear definition and relations to existing concepts. The SIMC System supports Catherine with auto-complete suggestions for parameters and relations. When the new SIMC is ready and automatically checked for consistency, Catherine publishes the model configuration, thus making it available to other users.

2.2.3 Simulation system model builder

This section describes a use case for a Simulation system model builder.

Use case 4: Maintain simulation system model

Liam is a Simulation system model builder for a specific simulation system. He has extended the properties of the entity 'human' of his simulation system with a parameter describing the emotional state. Liam has to update the model of the entity 'human" as stored in the simulation system model repository of the SIMC System. This will allow the SIMC System to use the concept of emotion from now on when converting a SIMC to a SMC for Liam's simulation system.

Interaction
Liam logs into the SIMC System and opens the simulation system model for his simulation system. He is now able to add the new "emotion" concept to the model of the entity 'human' of the simulation system model. When Liam starts typing, the SIMC System supports Liam by suggesting existing concepts relevant to 'emotional state', as defined in other simulation system models or SIMCs. Liam reads the accompanying definitions to see if a suggestion matches his new concept. If there is no suitable match, he can add a new concept "emotion" and specify any relations to existing concepts, such as behavioral aspects (e.g. facial expression, tone of voice). Liam also provides a definition of the concept "emotion" so that other users of the SIMC System may decide to include this concept in their simulation system models or SIMCs. Liam then publishes this new version of the simulation system model. Based on the relations defined in this model the SIMC System is able to relate the new concept to existing concepts and automatically take it into account when converting SIMCs into SMCs for that simulation system.

3 Modeling Approach

3.1 Introduction to OWL

OWL is a language for modeling ontologies in such a way that Description Logics can be applied to check for consistency and infer entity properties based on relations between entities. Originally it was developed for providing a way to represent knowledge on web content, as part of the Semantic Web. Soon after, OWL was applied to many other domains where complex ontologies are useful (e.g. health care). The current version of OWL is OWL 2, which was published in 2012 [5]. OWL 2 comes in a number of different language profiles; in this study, we are using the OWL Full profile.

The main building blocks of OWL are:
- *classes*, concepts that can be structured in a hierarchy and support multiple inheritance.
- *individuals*, instances of classes, cannot be structured in a hierarchy.
- *object properties*, relations that describe a property of two individuals.
- *data properties*, relations that describe a property between an individual and literal values such as an integer, string, double, etc.

Besides these building blocks, OWL contains a number of predefined concepts. Examples of these concepts are references to spaces (domain, range), relation properties (functional, transitive, symmetrical), cardinality (exactly, some), set comparators (disjoint, subset) and many more.

Let's look at an example of a definition of (a small part of) a simulation model configuration in OWL, focusing on a "CV90" armored vehicle. Structuring concepts in a class hierarchy in OWL is somewhat different than in, for instance, object-oriented programming languages, which are for the most part single-inheritance oriented. OWL is much more flexible and open ended in this respect. First, we'll introduce a number of base concepts as classes (note that we use the functional syntax, one of the many syntax-variants available for OWL, here for clarity):

```
SubClassOf(Vehicle Thing)
SubClassOf(ArmoredVehicle Vehicle)
SubClassOf(WheeledVehicle Vehicle)
SubClassOf(TrackedVehicle Vehicle)
SubClassOf(InfantryFightingVehicle Vehicle)
SubClassOf(MainBattleTank Vehicle)
```

`Thing` is the default root of any class hierarchy in OWL. In OWL, restrictions and exclusions have to be stated explicitly. This is because of a design choice in OWL known as the *open-world* assumption [6], combined with the fact that in OWL names are not considered unique by default (since they might be different names for the same concept). Therefore, one often has to explicitly state that classes and/or individuals are disjoint. In our example ontology, we want `InfantryFightingVehicle` and `MainBattleTank` to be exclusive concepts, but we will allow vehicles with both tracks and wheels.

```
DisjointClasses(InfantryFightingVehicle MainBattleTank)
```

Now we can define our individual, a `CV90`, by asserting that it belongs to one or more classes.

```
ClassAssertion(CV90 InfantryFightingVehicle)
ClassAssertion(CV90 TrackedVehicle)
ClassAssertion(CV90 ArmoredVehicle)
```

As said, OWL does not assume that a different name implies a different individual. Often, if this cannot be inferred from other assertions, it has to be stated explicitly.

```
DifferentIndividuals(Bradley CV90)
SameIndividual(CombatVehicle90 CV90)
```

Another point to note is that individuals cannot be hierarchically structured, instead they exist on the same level. In practice, this means that if we at a later point want to discern the different editions of the CV90 as individuals, e.g. CV9035NL, we might move the CV90 concept from the individual to the class structure.

We have seen how to structure the concepts in our simulation model as a hierarchy. Additional structuring can be introduced by defining relations: relations between concepts and relations of concepts with data. For this, OWL provides the object and data properties building blocks.

```
ObjectPropertyAssertion(hasWeaponSystem CV90 BushmasterIII)
```

Here we state that the `CV90` individual has a relation `hasWeaponSystem` with the individual `BushmasterIII` (its main weapon system). We can also state negative relations, e.g. explicitly stating which systems the CV90 does not have, and place restrictions on the relation, for instance:

```
ObjectPropertyRange(hasWeaponSystem WeaponSystem)
```

denoting that the range of the relation `hasWeaponSystem` is restricted to individuals that have been asserted to be of the class `WeaponSystem`.

Data properties, i.e. relations between individuals and data values, follow a similar structure.

```
DataPropertyAssertion(width CV90 "3.1")
```

Similar to object properties, we can further refine data properties, by, for instance, restricting the domain and range of the `width` property. Specifically for data properties, we can also assert what type of data they require (e.g. primitive types such as integer, floating-point number, etc.).

Everything stated so far is known as the *asserted* view on the ontology. However, by employing an OWL *reasoner*, an *inferred* view on the ontology can be derived. One of the main uses of such a reasoner is to check whether the asserted ontology is consistent. Furthermore, new information can be introduced in the inferred view, for instance a class X can be inferred to be a subclass of class Y, even though this was not explicitly asserted.

For a more in-depth introduction into OWL, many tutorials exist online. A good starting point is [7]. There are a number of generic ontology editors available with support for OWL. A popular and freely available tool, which was also used in this study, is Protégé [8]. Reasoners also come in many variants, differing in inference speed, capabilities and maturity. In this study, we used Pellet.

3.2 Modeling layers

Simulation model configuration data can be described at different levels of abstraction. The Object Management Group (OMG) defines a four-layered architecture for its modeling standards and we follow a similar scheme in defining the different levels of abstractions in describing simulation model configurations. The OMG has defined the following four layers:

Layer	Description
M3	Meta-metamodel layer
M2	Metamodel layer (metamodel that conforms to the meta-metamodel)
M1	Model layer (model that conforms to the metamodel)
M0	Instance layer (an instance of a model)

The focus of SIMC is on layers M0, M1 and M2, and layer M3 is considered out of scope for this paper. Furthermore, we introduce a differentiation between layers describing the simulation system *independent* model configurations and the simulation system *specific* configurations. The layers M1 and M0 describe a generic world representations and the layers S1 and S0 describe the same concepts at the same level of abstraction but from the point of view of a specific simulation system (see also Figure 3):

Layer	Description
M2	SIMC language and concepts definition
M1	Generic simulation model configuration
M0	Scenario specific model configuration with scenario specific parameter values
S1	Simulation system specific, but scenario independent model configuration
S0	Simulation system and scenario specific model configuration

The lowest simulation system independent layer M0 contains most of the actual configuration data values for a simulation model, called instances. Note that in OWL instances are called individual. At M0, we would define for example, our CV90 with all its specific properties, e.g. maximum speed of 70 km/hours, its dimensions, and a weight of 35 tonnes. These specific values are part of the simulation model configuration and used when simulating an entity of this type.

The M1 layer contains a description of the instances at the M0 layer. The descriptions at this abstraction level define for example the concept *InfantryFightingVehicle* (i.e. in this layer, OWL classes are defined). The relationship between the

M0 and M1 layer is that the elements in the M1 layer are a classification of the instances in the M0 layer and specify shared properties of each instance.

The elements in the M1 layer are in turn instances of the concepts defined in the M2 layer, such as class, attribute, and relationship. The elements in the M2 layer provide the language concepts for the elements in the M1 layer, and together form the SIMC language.

The simulation system-specific levels S1 and S0 are on the same modeling level as their generic counterparts M1 and M0, but contain the specific structure and concepts used in the simulation system. Both M0-1 and S0-1 are described in SIMC and may reference the same concepts/ontology entities. The resulting connected ontologies allow for a mapping between parameter values from M0 and S0 which can be used to translate SIMCs to SMCs. Note however, that a model configuration description defined in S0 is not equal to a SMC. S0 instances are described using SIMC (i.e. they are represented in an ontology) whereas SMCs are described using a simulation system specific file format.

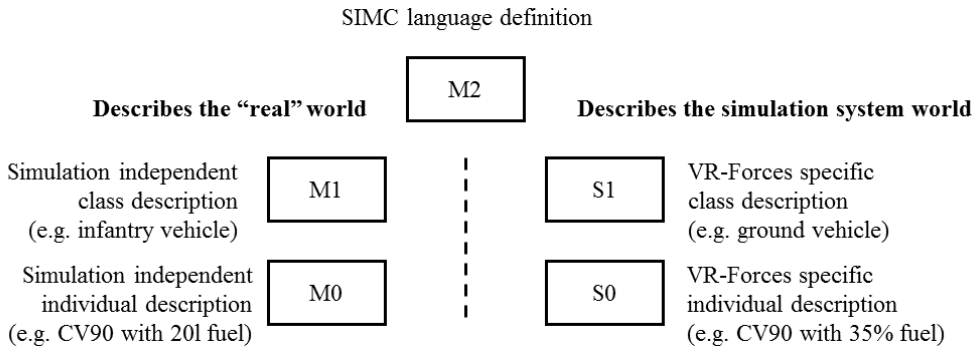

Figure 3: Modeling layers for SIMC.

4 Case Study

To explore the SIMC concept and the strengths and weaknesses of using OWL, a case study has been performed using a simple prototype implementation of the SIMC System. The case study is based on use case 2 described in section 2.2, where two scenario builders of different simulation systems want to exchange model configurations. The next sub-sections explains the set-up of the case study and the prototype. The last subsection gives a more detailed description of the SIMC ontology models.

4.1 Description

In the case study model specific parameters defined in a scenario for one simulation system are to be reused in a second simulation system as illustrated in Figure 4.

Figure 4: VR-Forces and JROADS.

The first simulation system is VR-Forces, a Computer Generated Forces simulation system from VT MAK [2]. The second system is JROADS, a simulation system for air defense systems developed by TNO [9]. These two simulation systems have been selected based on their availability, similarity in entity types, and accessibility of the underlying model configuration files. The model specific parameters that are to be reused describe a specific instance of a MiG-29 airplane. The end result would be an SMC for JROADS describing the MiG-29 in such a way that its behavior and appearance are

as similar as possible to the original MiG-29 defined in VR-Forces (taking into account that both systems have different underlying behavior models and graphics).

For the purpose of the case study only a part of the model configuration for the MiG-29 is used. The selection includes a representative set of different types of parameters among which:

- numeric values (e.g. weight, size, speed)
- boolean values (e.g. abilities)
- enumerations (e.g. forceside)
- positions (e.g. relative location of a weapon system)
- sub-classes (e.g. weapon systems)

The difficulty is that in general these parameters are represented very differently in the SIMC and the different SMCs. For example the *forceside* (i.e. the group to which a unit belongs in a military simulation) is represented both in the JROADS SMC and in the SIMC as an enumeration. However the possible values in both configurations are different:

Language	**Possible Values**
JROADS SMC	'Red', 'Blue', and 'Yellow'
SIMC	'Civil', 'Friendly', 'Hostile', 'Neutral', and 'Unknown'

4.2 SIMC System prototype

The use case asks for the transformation of a VR-Forces specific model configuration of a MiG-29 individual with custom parameters to a JROADS specific model configuration representing the same individual with the same custom parameters. In the case study the focus has been on the development of the different modelling layers required for this translation as described in section 3.2. We have studied the (system specific) model configurations for the MiG-29 in VR-Forces and JROADS and populated the M1 and M0 layers with an OWL model that represent a MiG-29 in a simulation system independent way, modelling the real-world as closely as possible. Next we populated the S1 layer with an OWL model describing a generic MiG-29 using JROADS specific structures and concepts. An extension application for the semantic reasoner along with several converters was developed to combine the information from these system independent and system specific model layers and to populate the S0 layer. The S0 layer describes the JROADS specific MiG-29 individual with custom parameters. The information in the S0 layer is still defined in OWL and cannot be interpreted by JROADS. The final step therefore was to develop converters to translate the information in the S0 layer to a JROADS specific format. Figure 5 provides an overview of the components of the prototype which are described in more detail below.

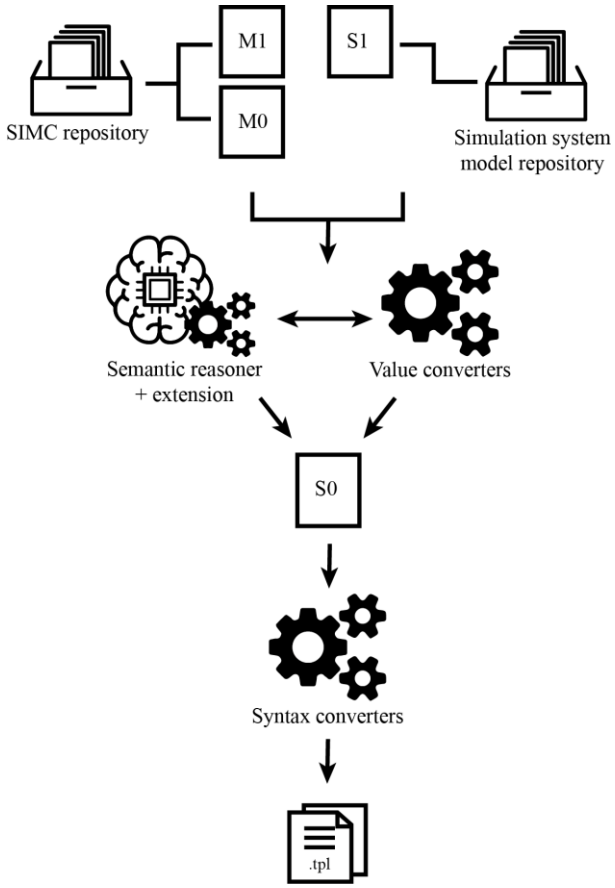

Figure 5: Overview of SIMC prototype implementation.

SIMC repository

To fill the SIMC repository the different modeling layers (M1, M0) have been populated by hand to model (parts) of the concept "MiG-29" in SIMC:

- **M1**: describes a generic MiG-29.
- **M0**: describes a specific MiG-29 individual.

Simulation system model repository

The information in the Simulation system model repository describes the semantics of JROADS SMCs. In the prototype only selected parameters for a MiG-29 are described by manually creating an S1 ontology. This is done in a generic way such that similar parameters for other entities can be covered as well.

Semantic reasoner and extension application

The semantic reasoner can be used to infer and create new individuals or expand present individuals based on existing ontologies. Based on the S1 ontology for a generic MiG-29 in JROADS it can create an S0 ontology for a specific MiG-29 individual for JROADS, listing all the required settings (e.g. max. speed). However, the current version of OWL and accompanying semantic reasoner are not able to create a fully instantiated S0 model that lists the actual values of the settings (e.g. max. speed = 2400 km/h) because they cannot handle unit conversions or simulation system specific enumeration mappings. Therefore we have developed an extension application that follows the references described in the S1 ontology in order to find the actual values for the settings in the M1 (generic MiG-29 settings) and M0 (specific MiG-29 individual settings) ontologies.

Value converters

The extension application crawls through the different model layers to find the appropriate values for all the settings in the S0 model. When it finds a value in, for example the M0 ontology, it might need to convert this value into an appropriate

format used by JROADS. This may include transformations like converting between metric units or changing the syntax. For this conversion the application calls upon generically defined converter functions.

File syntax converters
The JROADS specific S0 ontology of the MiG-29 cannot be interpreted directly by JROADS. A JROADS specific SMC converter translates the S0 ontology to JROADS specific configuration files.

4.3 Examples of the ontology model

In this section we elaborate on the structure of the OWL ontology by giving two examples. The figures are created using Protégé [8]. First we look at the simulation independent ontology in SIMC and then we show parts of the corresponding simulation specific ontology for JROADS.

Figure 6 shows part of the ontology in SIMC, where we can see several defined classes and their sub-class structure. The separation of information into classes allows us to specify different types of information at different locations, but still grouped together. An example of this is the separation of the physical information and configuration information. This allows for a property like 'mass' to be modeled on all physical objects.

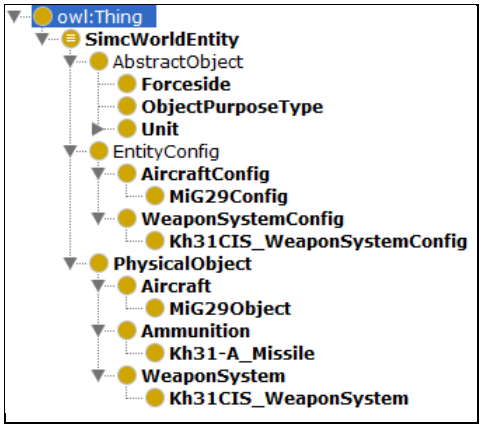

Figure 6: Snippet of class structure of SIMC ontology.

When we inspect the class *MiG29Config* in Figure 7 we can view several different properties being asserted at class level. It is clear that every individual belonging to the *MiG29Config* class has the property *hasDefaultForceside* with an individual *forcesideHostile*.

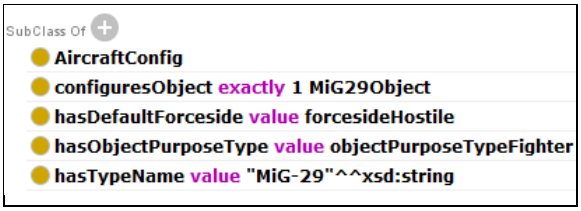

Figure 7: Snippet of class properties of *MiG29Config*.

Figure 8 and Figure 9 show the information in more detail. The *Forceside* class is asserted to have exactly one *forcesideValue* property that has a string range. Also the class consists of exactly five individuals. The *forcesideHostile* individual has a specific value for the *forcesideValue* property.

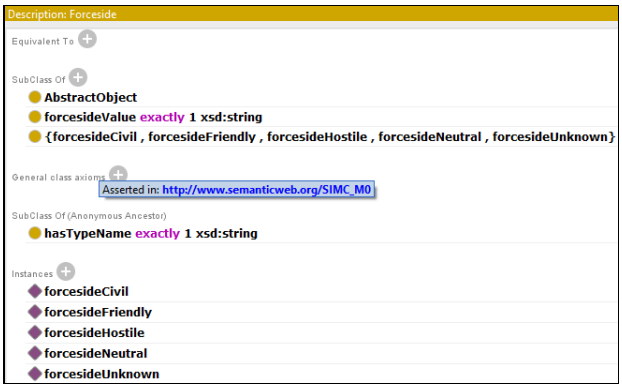

Figure 8: Forceside class.

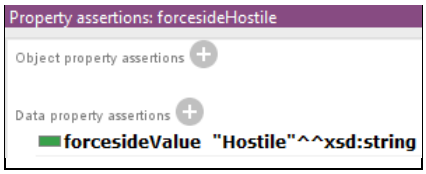

Figure 9: forcesideHostile individual.

In the JROADS specific ontology in Figure 10 the (S1) class *jroadsMiG29* is related to the (M1) class *MiG29Config* by the relation *derivedFromSimcWorldEntity*. The S1 layer captures the JROADS configuration structure and links the properties needed in the configuration to the concepts in the M1 layer.

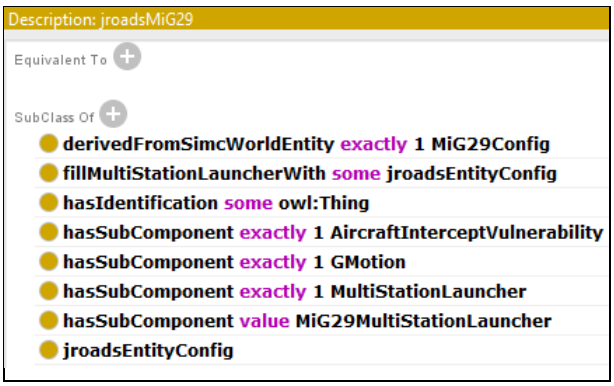

Figure 10: Snippet of JROADS simulator ontology.

5 Discussion

In the previous sections we explained the objectives, structure, and content for SIMC. The common focus is to build a SIMC System that uses the SIMC language as data model. The general usefulness of an OWL ontology as data model compared with some database system is the ability for a machine to be able to infer extra information on the data, thus enriching the available information on a basic level. Also, the specification of the data model is in a common and clear language. The general downside is that currently not all information can be specified easily and extra effort is needed to create such an ontology (see subsections below). Therefore, the added benefit of being able to share and reason about a data model should be carefully considered. During the implementation of the ontology and the extension application that uses it as its data model, we identified several points of interest and issues. These are discussed in the following sections along with recommended actions.

5.1 Open-world assumption

As mentioned earlier in section 3, the standard reasoning for OWL is based on the open-world assumption paradigm. The most important consequence is that all information in the data model based on the OWL ontology can only be information that is deductible with certainty. We have found that this assumption is not advantageous when one tries to capture a data model that is a closed-world, such as the data model of the simulation systems used in the case study. Changing the example from the OWL introduction a bit, we can define

```
SubClassOf(Bushmaster WeaponSystem)
ClassAssertion(BushmasterIII Bushmaster)
ObjectPropertyAssertion(hasWeaponSystem CV90 Bushmaster).
```

These state that a *CV90* always has a relation *hasWeaponSystem* with an individual of type *WeaponSystem*. We also defined one individual, *BushmasterIII*, of the type *WeaponSystem*. When the ontology is queried for what weapon a *CV90* has, the answer is unknown, even though the *BushmasterIII* is the only individual with type *WeaponSystem*. When some system uses the closed-world assumption, that system would have been able to answer with *BushmasterIII*. However, an open-world assumption system assumes that there might very well be other individuals of the type *Bushmaster*. In order to answer a closed-world question, or create an ontology that is closed, one can add statements to the ontology such that the reasoner can infer that the world is closed. In this example we should add a statement that the class *WeaponSystem* is equivalent to the group of individuals consisting of the *BushMasterIII*. When more individuals and classes exist, also statements on their disjointness or difference should be made.

When modeling a situation that is closed-world, such as the limited set of simulation objects available for a specific simulator, a choice has to be made. Either one closes the world with extra closing statements or one simply does not close the world. The first induces more inferred logic which the reasoner will verify in an inefficient way. The second option only allows to obtain information that is directly derivable from the input information. This makes the function of the reasoner often obsolete, as the desired information is already present in the input data.

We recommend to choose an ontology that uses an open or closed world assumption corresponding to the setting of the domain. A domain where all information is known corresponds well to the closed-world assumption. A domain where identification and determination of individuals is a main issue or not all information is explicitly known, adheres to the open-world assumption.

5.2 Relations are not semantic

The backbone of the semantics in the OWL language are the relations on the individuals and classes. These relations connect classes, give individuals properties or specify datatypes for properties. Together they explain the structure and properties of the classes and individuals in a set of basic terms and concepts known to the reasoner. This makes the individuals and classes semantic objects: the description logic and ontology together enable a reasoner to infer new information on the classes and individuals in the created ontology.

We can give a simple example of the above when we add a new individual to the OWL introduction ontology. For that individual we can assert that it does not have a *hasWeaponSystem* relation. Then the reasoner infers that this new individual is different from the *CV90*, since the latter does have a *hasWeaponSystem* relation.

However, in the current version of OWL, relations cannot be specified as an object or subject of another relation. This severely limits the capabilities of reasoning on the properties of relations. For any application that utilizes the relations of an OWL ontology, this means that the application needs to understand what the relations mean. Only then can an application interpret the information that is related to the relations. Suppose an application is interested in all information in the ontology that is related to the military. It is convenient if we could define a new data property *isMilitaryRelated* and let every relation have this property. This way, the application needs to semantically understand the *isMilitaryRelated* relation and gather all the information using this relation only. Since this is not possible, the application has to semantically understand more relations, for example *hasWeaponSystem*, *hasArmour*, *hasAmmunition*, *unitOf*, etc. This is why we claim that relations are not semantic.

We conclude that some general relational concepts that capture the semantics of a group of relations cannot be modeled. Therefore, we recommend that further investigations in this problem are made and that eventually the standard OWL model is altered to cope with this issue.

5.3 Classes and individuals

The distinction between classes and individuals is widely used and fits the task of modeling a part of the real world. However, it is important to realize that OWL reasoners only infer and verify information on the level of individuals. Classes simply enable the modeler to add information to groups of individuals. This implies that no matter how one models the classes in an OWL ontology, the inferred information on the individuals can still be the same. Therefore, it is unclear what the best approach is to model the classes of a number of individuals with several different and several common properties. One approach is to define a class for every property and use multiple inheritance for subclasses and individuals.

This issue becomes more complicated when we consider creating classes with fixed specified values for data properties. We encounter this problem when modeling the weapon system of a *MiG29*. First we create classes and subclasses to capture the information that simulators need of weapon systems. Then we relate the *MiG29* class to a weapon system. The problem is that we need to add information on the location of the weapon system relative to the *MiG29*, since some simulators need that information. This information can change per *MiG29* individual, although large groups of individuals of *MiG29* share the same relative location. Adding this information to either class (*MiG29* or the specific weapon system) implies that new classes have to be made each time the relative location of the weapon system changes. It is better to add this information to the relation itself. This way, any combination of *MiG29* and weapon system individuals share a specific relation where this information is stored. This is not possible, since relations are not semantic.

We can only recommend to evaluate and decide on the best modeling approach for subclasses and individuals, and apply this consistently.

5.4 Reasoner standards

There are many different reasoners that can check the consistency and infer information on an OWL ontology. The difference between reasoners can be quite large, impacting performance (speed) as well as the amount of inferred information. As a user of OWL it is unclear what each reasoner is good at or in what type of model structure the reasoner performs best. Although there are some references for this problem, see [10], they often have very specific recommendations.

Since there are no standards for reasoners, searching for one that suits the needs of the application can be a tedious task. We recommend that an accessible, regularly updated source of information is made on the status of all reasoners. This had to include a clear list of (high level) functionalities of the reasoner.

5.5 Default values

Currently, OWL does not support default values. In ontologies with the open-world assumption it is desirable to be able to let specific individuals overwrite a property. This problem is often solved by introducing new relations and asserting extra information on the individuals. However, that does not make the reasoner infer information on those relations, such that the value of the relations mimic a default relation.

For our extension application we choose to model the default relation as a new relation that is a subrelation of the ‘normal’ relation. This way the application can view the relation and subrelation as a whole, reading the default value from the subrelation when needed.

5.6 Quantities

In an OWL ontology a quantity can be specified by custom or built-in datatypes. However, the reasoner cannot infer information on the custom datatypes. This is a known issue, elaborated in e.g. [11], where the main issues of the dimensionality, units and reasoner abilities are discussed. For applications that do not need to reason on the units of the data, a simple annotation could suffice, letting a human reader know what quantities and measurement units are involved.

However, measurement units can differ for different simulators. Therefore, it is important that every data property has a specified measurement unit. We were able to specify the different units in OWL using additional properties. Also we moved the conversion of units to the extension application, removing it as one of the tasks expected from the reasoner (as mentioned in section 4.3). This solution has been implemented generically, thus making it an inherent assumption and property of the SIMC System. Therefore, any extension application can read the values and their unit, followed by calling a conversion method.

5.7 Logical properties

In the current version of OWL there is no possibility to specify logical relations that a reasoner can verify or infer information on. Therefore, a logical relation, such as ‘smaller than’, cannot be reasoned on. Also more mathematical functions such as ‘sum’ or ‘product’ cannot be specified. For our extension application we did not need any of these constraints.

The previous sections on default values, quantities and logical properties all involved issues on functionality that is not yet included in OWL or the reasoner. We therefore recommend further study into these issues and eventually to extend the OWL language in order to solve the issues.

5.8 Over and under specification

Due to the open-world assumption of OWL, there is a thin line between being able to infer information or not. To be able to infer information requires specific conditions to be set on various properties, classes and individuals. When modeling an ontology with the closed-world assumption, it is beneficial to assert a minimal amount of information, with a maximal amount of information to be inferred. Over specification leads to high complexity, for users and the reasoner. Under specification causes less information to be inferred, making the ontology less useful. Avoiding over and under specification is complicated to achieve.

The risk of over-specifying an ontology is high. This is especially imminent when properties imply the same piece of information, or a piece of information can be specified at several locations, eventually inferring the same thing. An example is the possibility to specify a class as a disjoint union, which implies the subclasses are disjoint. Then those subclasses do not have to be specified as disjoint. Another example is the domain and range specification of relations. When adding an individual to the ontology, in a closed-world setting, it is known what entity this individual is. Therefore, the reasoner does not need the domain and range of relations to infer new information on the type of class of the individual, since that class is already known. A last example is that the disjointness of classes implies the disjointness of individuals.

In order to overcome this issue we recommend investigate in the automatic creation of ontologies where a system can strictly follow design decisions. Of course, a clean interface can be built in order to let the human interact with the ontology and a machine check whether these decisions are satisfied.

5.9 Final remarks

The open-world assumption and the conclusion that relations are not semantic are fundamental to the major problems discussed here. The open-world assumption limits the potential value of a reasoner, since in a closed-world setting reasoning is limited. The open-world assumption also easily leads to over-specification of the ontology. Other ontologies may be able to provide more satisfying solutions to the issues address in this paper, depending on their specific mechanics. For example, Frame ontologies assume a closed-world, which may prevent some of the issues of OWL, although other problems might arise. For a comparison of OWL and Frame ontologies, see [12].

6 Summary and Conclusions

The configuration of simulation models is currently simulation system dependent and simulation models all have model specific parameters. These parameters determine to a large extent the resulting behavior of simulated entities, their interactions, and the effects of their interactions at run-time, and thus ultimately the outcome and validity of a simulation execution. Model parameters are therefore important. However, no standards exists to describe such parameters and their semantics, to exchange these between simulation systems, and to reference these from simulation scenarios.

This paper investigated the use of the Web Ontology Language (OWL) to describe simulation model parameters in a simulation system independent way. OWL was selected for its expressive power, although this feature introduces added complexity in capturing parameters and semantics. The language introduced in this paper, built on top of OWL, is denoted the Simulator Independent Model Configuration (SIMC) language, in which a layered structure is applied for the description of knowledge about model parameters. This paper described several use cases for a SIMC System and presented a case study to explore the system concept with the aim to learn about the strengths and weaknesses of using OWL for this purpose. A small prototype implementation of the system was developed with a simulation system independent description of model parameters of a MiG-29 airplane. These parameters were subsequently exchanged between two simulation systems: TNO JROADS and MaK VR-Forces. In performing this case study several issues in the use of OWL were identified, with recommended actions for most of them.

In conclusion, the case study shows that an ontology based and simulation system independent description of model parameters is feasible. It is possible to describe model parameters and their semantics in OWL, and exchange these between different simulation systems. However, the translation between a simulation system independent description of model parameters in OWL and a simulation system specific model configuration is mostly a system dependent implementation due to the variety in simulation systems. In addition, the use of OWL has several implications. Various issues and modeling approaches need to be taken into account when developing an OWL description. Most relevant are the adoption of the open-world assumption by OWL and non-semantic relations. It turned out that semantic reasoners are less valuable in a closed-world or in a situation where all parameter values are known, and this may often be the case for simulation model configurations. On the other hand, the use case demonstrated the strength of OWL, being the ability to describe information from multiple sources and create a shared ontology or vocabulary, in this instance an ontology for the concepts used by different simulation models and systems. The scope of the case study in this paper is relatively limited, but it is clear that a number of issues exist when using OWL. Some of the identified issues need follow-up in the OWL community and further research is needed to explore and resolve them. In addition, other ontology languages should be evaluated that work with the closed-world assumption.

Given the importance of simulation model configurations, our recommendations to SISO are:

C2SIM PDG/PSG:
Support the description of simulation model configurations in the executable scenario; i.e. adapt the MSDL Model element in the EquipmentItem element to support references to model configurations. Descriptions may use different schemas for different languages, e.g. for a Domain-specific language (DSL) or OWL.

C2SIM PDG/PSG:
Standardize matured descriptions of simulation model configurations so that simulation system providers can add support in their products.

GSD PDG/PSG:
Include the development of simulation model configurations in the scenario development process.

7 References

[1] SISO, “Guideline on Scenario Development for Simulation Environments”, SISO-GUIDE-006-201X (draft), 2016, SISO GSD PDG.

[2] VT MAK, https://www.mak.com.
[3] SISO, “Final Report for the Architecture Neutral Data Exchange Model Study Group”, SISO-REF-058-2015, 5 January 2015.
[4] IEEE, “IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP)”, IEEE Standard 1730-2010.
[5] Boris Motik, Peter F. Patel-Schneider, Bijan Parsia. “OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (Second Edition)”, W3C Recommendation, 11 Dec 2012.
[6] Open-world assumption, Wikipedia, https://en.wikipedia.org/wiki/Open-world_assumption
[7] OWL 2 Web Ontology Language Primer (Second Edition), https://www.w3.org/TR/owl2-primer.
[8] Musen, M.A. The Protégé project: A look back and a look forward. AI Matters. Association of Computing Machinery Specific Interest Group in Artificial Intelligence, 1(4), June 2015.
[9] TNO JROADS, https://www.tno.nl/en/focus-areas/defence-safety-security/missions-operations/jroads.
[10] Bock, Jürgen, et al. "Benchmarking OWL reasoners." ARea2008-Workshop on Advancing Reasoning on the Web: Scalability and Commonsense. Tenerife, 2008.
[11] Parsia, Bijan, and Michael Smith. "Quantities in OWL." Clark & Parsia LLC, Washington, DC (2008).
[12] Wang, Hai H., et al. "Frames and OWL side by side." Presentation Abstracts. 2006.

8 Author Biographies

FREEK VAN WERMESKERKEN is a scientist at TNO since 2015 and has a MSc degree in Mathematics from the Vrije Universiteit Amsterdam. He specializes in optimization and modeling dynamical systems. His work focuses on creating models and algorithms to aid the field of safety and security on topics of analysis and decision support.

GWEN FERDINANDUS is a research consultant in the Modeling, Simulation, and Gaming department at TNO. She holds a MSc degree Technical Artificial Intelligence from Utrecht University. Her work focusses on the application of knowledge and innovation to the specific simulation challenges of the customer. She currently works on a broad range of application domains such as human behavior modeling, intelligence augmentation, and the simulation of cyber effects.

TOM VAN DEN BERG is a senior scientist in the Modeling, Simulation and Gaming department at TNO, The Netherlands. He holds an M.Sc. degree in Mathematics and Computing Science from Delft Technical University and has over 25 years of experience in distributed operating systems, database systems, and simulation systems. His research area includes simulation systems engineering, distributed simulation architectures, systems of systems, and concept development & experimentation.

RUBEN SMELIK is a scientist at TNO since 2007. He holds a MSc degree in computer science from Twente University. He earned a PhD degree from Delft University of Technology based on his thesis on the automatic creation of 3D virtual worlds. His current work focuses on innovations in the field of automated synthetic environment modeling for military simulation applications.

KAREL VAN DEN BOSCH is senior researcher for TNO. He holds a PhD degree in social sciences from the University of Nijmegen. At TNO he manages (inter)national research projects on the training of complex cognitive tasks (e.g., decision making, crisis management). The objective of his current work is to make simulation-based training more effective by using cognitive software agents (e.g., agents playing the role of team mate, adversary, or instructor).

HENK HENDERSON is a senior scientist in the Modeling, Simulation and Gaming department at TNO, The Netherlands. He holds an M.Sc. degree in Mathematics and Computing Science from Delft Technical University and has over 25 years of experience. His work focuses on distributed operating systems and the software engineering aspects of modeling & simulation.