Security Informatics
a SpringerOpen Journal

RESEARCH Open Access

Generating test cases for marine safety and security scenarios: a composition framework

Hamed Yaghoubi Shahir[1*], Uwe Glässer[1], Roozbeh Farahbod[2*], Piper Jackson[1] and Hans Wehn[3]

* Correspondence: syaghoub@sfu.ca; roozbeh.farahbod@sap.com
[1]Software Technology Lab, Computing Science, Simon Fraser University, B.C., Canada
[2]SAP Research, Karlsruhe, Germany
Full list of author information is available at the end of the article

Abstract

In this paper we address the problem of testing complex computer models for infrastructure protection and emergency response based on detailed and realistic application scenarios using advanced computational methods and tools. Specifically, we focus here on testing situation analysis decision support models for marine safety & security operations as a sample application domain. Arguably, methodical approaches for analyzing and validating situation analysis methods, decision support models, and information fusion algorithms require realistic *vignettes* that describe in great detail how a situation unfolds over time depending on initial configurations, dynamic environmental conditions and uncertain operational aspects. Meaningful results from simulation runs require appropriate test cases, the production of which is in itself a complex activity. To simplify this task, we introduce here the conceptual design of a Vignette Generator that has been developed and tested in an industrial research project. We also propose a framework for composing vignettes from reusable vignette elements together with a formal representation for vignettes using the Abstract State Machine method and illustrate the approach by means of various practical examples.

Keywords: Marine Safety & Security, Test Case Generation, Infrastructure Protection, Emergency Response, Abstract State Machines

1 Introduction

Infrastructure protection and emergency response scenarios routinely call for intelligent coordination and management of multiple mobile resources, often operating in vast geographical environments. For instance, mobile platforms including patrol and cargo airplanes, rescue helicopters, unmanned aerial vehicles (UAV), satellites, ground vehicles, and search & rescue (SAR) vessels are deployed for the gathering of information that is crucial for situation analysis and for the transportation of persons and supplies. Typical scenarios include: intervention by border control services in illegal activities, such as smuggling operations and piracy; routine surveillance and SAR missions conducted by coast guard services; as well as emergency services and first responders in disaster relief operations, for instance, after a major earthquake with a devastating tsunami or a catastrophic oil spill. Marine safety & security is critical for Canada to address the vulnerability of its sea lanes, ports and harbors to a variety of threats and illegal activities. Scarce surveillance and tracking capabilities make it difficult to keep track of all marine traffic across the length of Canada's coastline, which totals over 243,000 kilometres [1]. A coordinated response to emergency situations,

like many safety and security operations, frequently involves a number of mobile agents, cooperatively performing missions and reacting to events that are distributed in time and space.

Situation Analysis (SA) is viewed as a process to provide and maintain a state of situation awareness for the decision maker [2]. Situation awareness is essential for decision-making activities: it concerns our perception of the elements in the environment, the comprehension of their meaning, and the projection of their status in the near future [3]. Computational models of situation analysis processes are in many cases of a distributed nature, being composed of multiple autonomously operating agents that each react in an asynchronous manner to discrete events distributed in space and time. Agents cooperate in developing a global understanding of a situation as it unfolds by exchanging information related to their local perception of events. Computer models support risk assessment and disaster response planning in the study of emergency preparedness by providing a practical alternative to real-world experiments, which in many cases are so costly and disruptive when performed on real situations that they do not provide a viable option.

This paper addresses the problem of testing complex computer models of infrastructure protection and emergency response situations based on detailed and realistic application scenarios in order to analyze and validate such models by means of advanced computational methods and tools. Although the primary focus in this paper is on marine safety & security operations [4,5], we contend that the same approach carries over to a much broader scope of application fields. This is due to two facts: *1)* the concepts proposed here build on general abstractions of distributed systems composed of multiple mobile agents; and *2)* the way use cases are formally specified as abstract models of *vignettes* [6] in terms of *state machines* [7].

Methodical approaches to analyzing and testing decision support systems, situation analysis methods, and information fusion algorithms require realistic vignettes describing in great detail how use cases unfold depending on initial conditions for the configuration of agents, environmental aspects, and operational aspects. For instance, this includes the type and placement of agents in the geographic environment, their physical and operational capabilities, and their anticipated trajectories; specifics about environmental conditions, including weather, daylight, and water currents; and background noise, such as the number and distribution of neutral entities like any unrelated marine traffic. Realistic scenarios may include dozens or even hundreds of agents and complex conditions to be considered for each individual instance of a given vignette. Generating vignettes is a complex activity, one that is utterly inefficient to do manually and thus benefits greatly from automatic or semi-automatic approaches with tool support. We present here the conceptual design of a vignette generator to address this problem.

This paper is structured as follows. Section 2 outlines the problem scope, explains fundamental concepts and also the main goals of generating vignettes. Section 3 explains the compositional framework proposed for generating and/or composing vignettes out of reusable vignette elements. Section 4 discusses the conceptual design, including the main requirements, and the architecture of the system. Section 5 illustrates the underlying formalization approach for vignettes based on the *abstract state machine* method by means of common examples. Section 6 discusses applications and benefits of the Vignette Generator. Section 7 concludes the paper.

2 Fundamental Concepts

This section introduces the problem scope, some basic terminology, vignette patterns, and fundamental concepts used for describing the Vignette Generator in subsequent sections.

2.1 Problem Scope

Detailed experimental studies of emergency scenarios by means of computer simulation and animation play a crucial role in the development of innovative ICT solutions for situation analysis and decision support [6,8-10]. Furthermore, they are gaining momentum as a viable alternative to performing experiments in a real-world setting because of the inevitable limitations of any large-scale real-world experiment. Scenarios explored in the work presented here deal with operations performed by Marine Security Operation Centres (MSOC) [11]. This includes SAR operations for passengers and vessels in distress, as well as the protection of sea lanes, ports and harbors against threats and illegal activities. The increasing volume of marine traffic [12] calls for advanced computer-based systems to support MSOC personnel in their daily missions by automating routine coordination tasks, and building on common surveillance technologies such as the Automated Identification System (AIS) [13]. For this purpose, it is essential to analyze scenarios, evaluate algorithms and assess the quality of solutions systematically by performing in-depth experimental studies based on realistic and meaningful test cases with a degree of detail beyond what could be done manually. The approach for the generation of test cases proposed here assumes interactive development, analysis and validation of complex vignettes for testing realistic scenarios.

2.2 Basic Terminology

We define the following basic terms to disambiguate the concepts of domain, scenario, vignette, vignette specification, and vignette element [6].

Domain Model

A *domain model* is a conceptual model of a given problem domain, defining the various types of entities that are relevant for this domain, their attributes, relationships, constraints, and behaviors.

Scenario

A *scenario* is a specific interpretation of a domain model for a given geographical area and time period. For instance, a marine safety & security scenario may refer to the Straight of Georgia between Vancouver Island and the mainland of British Columbia in the current time.

Vignette

A *vignette*, denoted by $\mathcal{V}$ in this paper, is a story embedded in a scenario. The story unfolds as a set of discrete events involving agents and the physical environment in which they operate, describing the distribution of agents and events in time and space. For instance, a smuggling operation in the Straight of Georgia, off the northern shore of Vancouver Island, can be described as a vignette embedded in a marine safety & security scenario.

Vignette Specification

A *vignette specification* is a precise and structured, static, text-based description of a vignette.

Vignette Element

A *vignette element*, denoted by $\mathcal{E}$ in this paper, is any identifiable part of a vignette specification.

Vignette Generator

The term *vignette generator* refers to a tool for generating vignette specifications based on vignettes.

A vignette specification generated by a vignette generator conceptually represents a *state machine model* that can be interpreted by a simulation environment to produce simulation runs.

2.3 Vignette Patterns

Vignette elements are basic building blocks for constructing vignettes and therefore vignette specifications. They usually have parameters that allow adjustment for a given target context. Vignette elements can be considered as *reusable patterns* in the context of marine safety & security. These four patterns serve as examples:

- *White Traffic Area Pattern*: In this pattern, there are a number of neutral agents (e.g., Fishing Boats) that are moving randomly within a specified area. Such agents are considered part of the background, and referred to as "white traffic" (see Figure 1(a)).
- *Routine Patrolling Pattern*: In this pattern, an agent with the ability to visually capture the environment (e.g., a Helicopter or UAV) moves within a predefined path and provides environmental information for other agents (see Figure 1(b)).

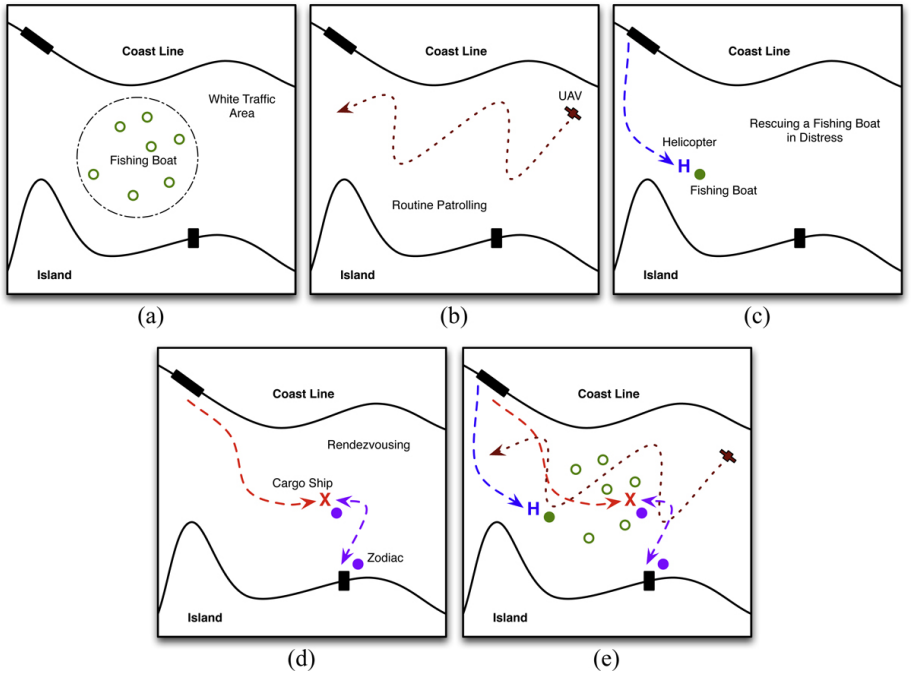

Figure 1 (a) White-Traffic Area Pattern, (b) Routine Patrolling Pattern, (c) Rescuing Pattern, (d) Rendezvousing Pattern, (e) Composite Vignette.

- *Rescuing Pattern*: In this pattern, there is an agent (e.g., a Fishing Boat) in a distress situation. Coast guard rescue team (e.g., a Helicopter) tries to rescue the people involved and also secure the trapped agent (see Figure 1(c)).
- *Rendezvousing Pattern*: This pattern involves a larger vessel, like a Cargo Ship, and a number of smaller boats, like Zodiacs. The bigger vessel stays at a fixed location (the Rendezvousing Point), while the smaller vessels go back and forth between the rendezvousing point and possibly a location close to the shoreline (the Beach Point). Figure 1(d) illustrates a rendezvousing pattern for a cargo ship and two zodiacs.
- Any combination of these vignette elements can be considered for generating a composite target vignette. Figure 1(e) is an example of a combination of all four vignette elements.

2.4 The Vignette Generator

As already mentioned, the Vignette Generator is a tool for developing well-defined vignettes and systematic generation of vignette specifications. Arguably, using a vignette generator has a number of decisive advantages as it makes vignette specification much *easier*, *more time efficient*, and *less error prone*. That is, users will be freed from the complicated and tedious task of describing low-level details and repeating routine activities. In particular, it allows the automatic generation of statistically relevant test cases as variations of vignette specifications that can be derived from a convenient high-level specification. The following simple example illustrates the role of the Vignette Generator for defining the White Traffic Area Pattern.

Example 1

We need to specify 100 Fishing Boats within a given area, each equipped with a Communication Device with certain properties. A traditional approach, would have to define each Fishing Boat separately as follows:

```
WhiteTrafficArea
 FishingBoat-1
    Position(X, Y) = (x-1, y-1)
    CommunicationDevice-1
       Type = link-11
       Range = 5000 m
       .
       .
       .
 FishingBoat-100
    Position(X, Y) = (x-100, y-100)
    CommunicationDevice-100
       Type = link-11
       Range = 3000 m
```

In our proposed approach and tool, the vignette element (White Traffic Area Pattern) is described as follows:

```
WhiteTrafficArea
 #number = 100
 FishingBoat
    Position(X,Y) = #random value within area A1
    CommunicationDevice
       Type = link-11
       Range = #random value between (2000 m) and (5000 m)
```

#: These elements and values are processed by the Vignette Generator engine.

Advantages

In the first specification, the details are constructed manually. In the second, the Vignette Generator will produce these details automatically. From the example, it is clear that the proposed approach has the following benefits.

1. Defining and understanding the second specification (7 lines) is much easier than the first one (501 lines).

2. Generating vignettes (or vignette elements) is more time efficient, specifically when the user needs to generate a number of vignette specifications based on a common pattern. This helps with statistical tests, which are necessary for evaluating algorithms.

3. Any error in the second specification is much easier to find and fix than in the first specification.

3. Vignette Composition Framework

In this section, we propose a framework for composing a new *Vignette Element Type* ($\mathcal{ET}$) from a set of existing vignette element types.

Initial Type Set ($\mathcal{I}$) is defined as a set of initial types which are the fundamental building blocks for composing new $\mathcal{ET}$s. Although $\mathcal{I}$ can be extended if required, its suggested elements are as follows:

$$\left\{ \begin{array}{c} \textit{Angle, Boolean, DateTime, DeltaTime, Energy, Length,} \\ \textit{Real, Speed, String, Temperature, Volume, Weight} \end{array} \right\} \in \mathcal{I}$$

Vignette Element Type ($\mathcal{ET}$) is considered as a building block for generating a *vignette* or another $\mathcal{ET}$. Each $\mathcal{ET}$ has *name*, *type*, and also a set of $\mathcal{ET}$s as its *attributes*.

Example 2

Before explaining our approach in detail, we illustrate the idea by means of a simplified $\mathcal{ET}$, the *CH-149 Helicopter*. *CH-149* is a helicopter which has the *attributes* listed below. For instance, *OpticalSensor* $\mathcal{ET}$ is one of its attributes that in turn identifies specific attributes for this sensor element.

```
CH-149: Helicopter
 Height: Length
 Width: Length
 Length: Length
```

```
FuelCapacity: Volume
CruisingVelocity: Speed
MaximumVelocity: Speed
Communication: CommunicationDevice
   Frequency: Real
   Range: Length
   Bandwidth: Real
OpticalSensor: SensorDevice
   SamplingPeriod: DeltaTime
   Range: DeltaTime
```

A generated composite $\mathcal{ET}$ can be stored in a repository named *Vignette Repository* ($\mathcal{R}$) for future use. Figure 2 illustrates a simple example of $\mathcal{R}$.

Vignette Element Types ($\mathcal{ET}$) and Vignette Elements ($\mathcal{E}$): We have defined Vignette Elements as identifiable parts of a vignette or vignette specification; also, they are considered as *instances* of Vignette Element Types. Thus, in order to include a Vignette Element in a vignette, we have to *instantiate* the appropriate $\mathcal{ET}$, and then *assemble* it on the target *vignette* (see Section 3.1 for details).

3.1 Composition Mechanisms

Composition mechanisms are proposed with the purpose of creating new $\mathcal{ET}$s and $\mathcal{E}$s. These mechanisms are categorized in two different sets. The first set, $\mathcal{ET}$-related mechanisms, is used to generate new composite Vignette Element Types from existing $\mathcal{ET}$s in $\mathcal{R}$. The second set, $\mathcal{ET}/\mathcal{E}$-related mechanisms, is used to generate Vignette Elements from appropriate $\mathcal{ET}$s.

- *$\mathcal{ET}$ Mechanisms*
  - *Extending*: Adding a Vignette Element Type *et'* to an existing element type *et* as its attribute.

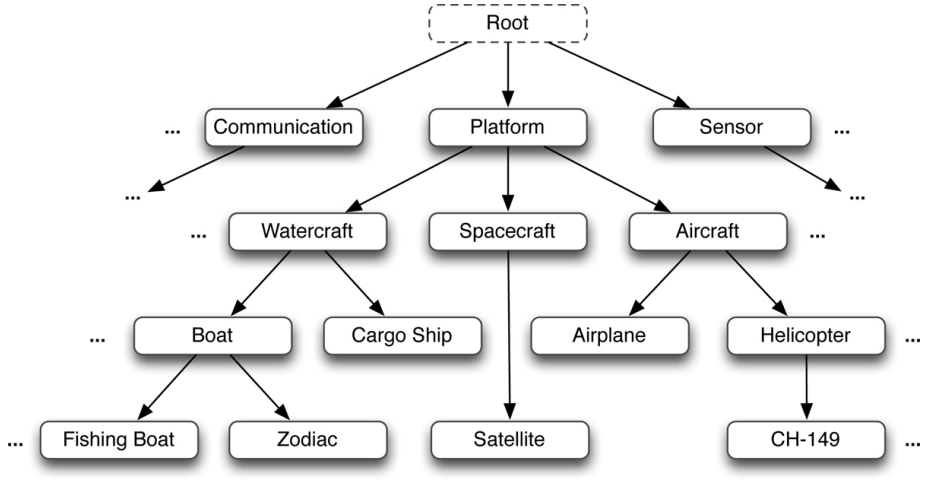

Figure 2 Example of the Element Repository ($\mathcal{R}$).

- *Initiating*: Creating a new Vignette Element Type *et'* as a sub-type of another element type *et*.
- *Cloning*: Cloning an existing Vignette Element Type *et* with a different name *et'*.
- *Tuning*: Adjusting the value of the attributes of an existing $\mathcal{ET}$.

- $\mathcal{ET}/\mathcal{E}$ *Mechanisms*
  - *Instantiating*: Creating an $\mathcal{E}$ as an instance of $\mathcal{ET}$, and *tuning* all of its attributes.
  - *Assembling*: Adding an instantiated $\mathcal{E}$ to the target vignette $\mathcal{V}$, and therefore vignette specification.

All in all, a Vignette $\mathcal{V}$ is a set of Vignette Elements $\mathcal{E}$s which are instances of Vignette Element Types $\mathcal{ET}$s (see Figure 3).

3.2 Relationships between Vignette Element Types

Two different types of relationships are defined among $\mathcal{ET}$s:

- *Has-a Relationship*: Each $\mathcal{ET}$ can have a number of $\mathcal{ET}$s as its *attributes*.
- *Is-a Relationship*: $\mathcal{ET}$s can have hierarchical relationships. In this way, each $\mathcal{ET}$ has a super-type (except for the root element). All sub-types of an $\mathcal{ET}$ inherit all

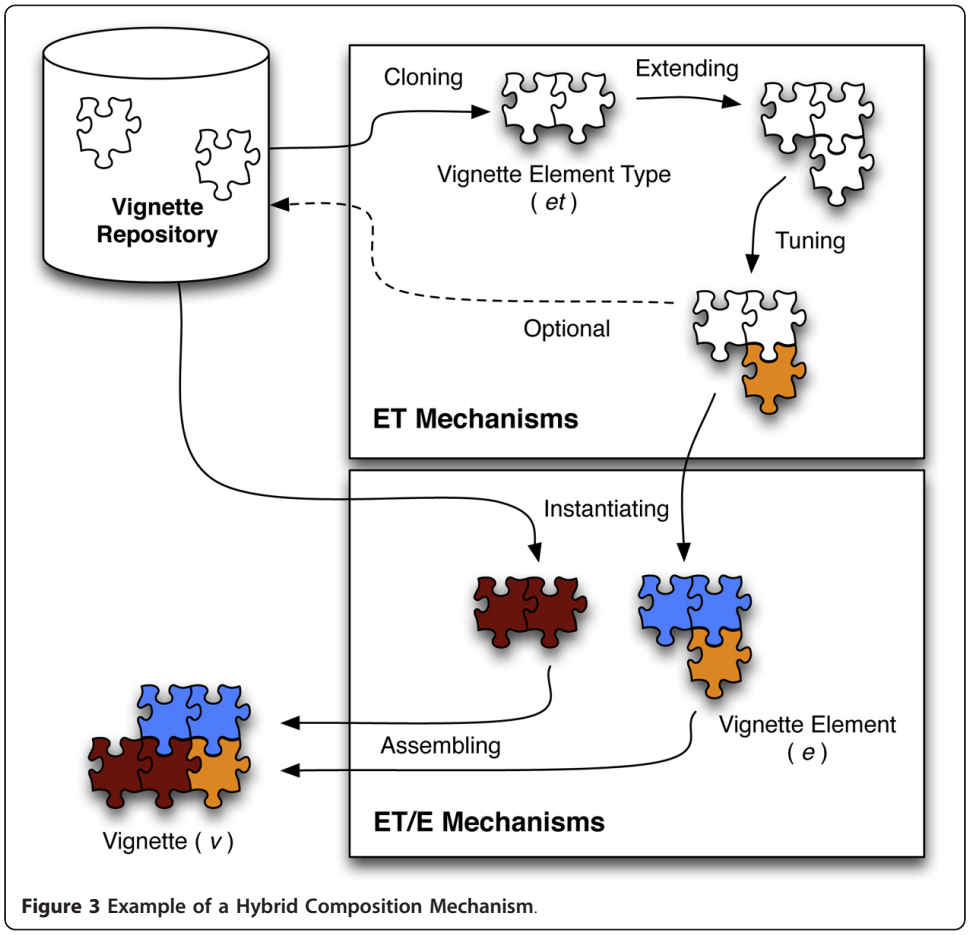

Figure 3 Example of a Hybrid Composition Mechanism.

has-a relationships from their super-type. However, they are able to override these inherited relationships. For example, if an $\mathcal{ET}$ (et_1) *has-a* $\mathcal{ET}$ (et_2), all sub-types of et_1 will have et_2. By this approach, we prevent redundancies in defining $\mathcal{ET}$s and their relationships. Moreover, it helps to maintain the $\mathcal{ET}$s in a more efficient way.

Since the number of Vignette Element Types can increase considerably over time, these relationships help in organizing them within the repository.

Example 3

The composition of a Routine Patrolling Pattern (Figure 1(b)) and Rendezvousing Pattern (Figure 1(d)) is illustrated here. By applying the following steps, we can generate the composed vignette. Assume that we already have *CargoShip*, *Zodiac*, and *UAV* $\mathcal{ET}$s in the repository.

1. *Initiate* Routine Patrolling Pattern $\mathcal{ET}$ as a subtype of *root* $\mathcal{ET}$.
2. *Extend* Routine Patrolling Pattern $\mathcal{ET}$ with a *UAV* $\mathcal{ET}$.
3. *Initiate* Rendezvousing Pattern $\mathcal{ET}$ as a subtype of *root* $\mathcal{ET}$.
4. *Extend* Rendezvousing Pattern $\mathcal{ET}$ with a *CargoShip* $\mathcal{ET}$.
5. *Extend* Rendezvousing Pattern $\mathcal{ET}$ with two *Zodiac* $\mathcal{ET}$s.
6. *Tune* the *trajectory* attributes of the *CargoShip* and *Zodiac* $\mathcal{ET}$s.
7. *Instantiate* Routine Patrolling Pattern $\mathcal{ET}$ and *Tune* all of its attributes to generate Routine Patrolling Pattern $\mathcal{E}$.
8. *Instantiate* Rendezvousing Pattern $\mathcal{ET}$ and *Tune* all of its attributes to generate Rendezvousing Pattern $\mathcal{E}$.
9. *Assemble* Routine Patrolling Pattern $\mathcal{E}$ and Rendezvousing Pattern $\mathcal{E}$ into a new current vignette ($\mathcal{V}$).

4. Conceptual Design

This section discusses the conceptual design of the Vignette Generator, including the system requirements and high-level architecture.

4.1 System Requirements

The requirements of the Vignette Generator outlined here were elicited and analyzed in part through discussion with domain experts. Afterward, high level requirements were identified and classified into different categories. For each category, the most significant lower level requirements were explored and discussed.

1. *Flexible support for a variety of vignettes*: The Vignette Generator should support definition of different vignette elements, with fixed or variable characteristics, that can be reused and combined in creating vignette specifications.
    a) The ability to easily specify a set of vignette elements implicitly by specifying a vignette element together with a set of parameters. The Vignette Generator should then automatically generate an explicit set.
        - Example: In the White Traffic Area Pattern (Figure 1(a)), the user can specify a single concrete Fishing Boat (an existing vignette element from the repository), and set a few parameters such as the number of boats desired and an area. Then the Vignette Generator produces a specification with that

many boats distributed over this area, moving about in a certain manner. The parameter varied is the initial trajectory of each boat (see Example 1).

b) The ability to aggregate, name, store, and retrieve previously defined vignette elements in a systematic way.

- Example: A new vignette element with some specific features and attributes can be given a name, e.g., CH-149 Helicopter, and stored in the repository. Later on, we should be able to reuse the stored element as is or extend it for other uses.

- Example: The White Traffic Area Pattern and the Rendezvousing Pattern (Figure 1) can be stored in the repository, and reused to compose a new complex vignette.

c) The ability to expand the repository of vignette elements over time which leads to the gradual generation of more and more complex vignette elements and vignettes.

2. *Support for non-determinism*: The user should be able to abstract away from the detailed configuration of vignette elements through non-deterministic value assignments to the attributes of vignette elements.

a) The ability to easily specify a set of vignette elements from which the generator picks one at random.

- Example: If Aircraft is specified, the generator randomly substitutes one of the concrete platforms that is a subtype of Aircraft, such as the CH-149 Helicopter (Figure 2).

- Example: In a traffic area that we have a number of platforms which have communication devices, the generator randomly assigns one of the concrete communication devices that is a subtype of Communication (Figure 2).

b) The ability to easily specify parameterized (unfixed) values (e.g., randomly, from a range or distribution) for attributes of vignette elements.

- Example: The value of Velocity attribute of CH-149 Helicopter should be selected randomly from a range between X and Y ($X \geq Velocity \geq Y$).

3. *Validation of high-level vignettes*: The Vignette Generator should allow validation of scenario descriptions against domain-specified and user-defined sets of constraints.

a) The ability to automatically check for errors, inconsistencies, policy and best-practice violations, and omissions.

- Example: The generator should issue warnings for integrity violations such as inconsistent behavior, e.g., ships traveling over land or the use of communication equipment that is incompatible with all others.

b) The ability to automatically check for user-defined constraints. The Vignette Generator should issue warnings for any violations.

- Example: The value of the Altitude attribute of all Aircraft should be between a minimum and a maximum value; otherwise, the system should report a warning. ($X \geq Altitude \geq Y$).

4. *Production of concrete vignette specifications for different simulation environments*: From every abstract vignette that complies with the requirements of a given simulation environment, the Vignette Generator should be able to generate a number of concrete vignette specifications in a specified format (e.g., XML) that can be

loaded in the simulation environment. To this end, the engine of the Vignette Generator should not depend on any specific simulation environment. This requirement is useful for generating vignettes for different simulators or even a simulator under construction.

5. *Easy to use Graphical User Interface (GUI)*: The Vignette Generator should come with an easy to use GUI, which supports the creation, modification, and instantiation of vignette elements.

4.2 System Architecture

To address the mentioned functional and non-functional requirements, we propose a two-layer architecture (see Figure 4).

- *Simulator Independent Layer*: This layer is responsible for composing a new vignette based on Vignette Element Types and Vignette Elements. The output is not executable and needs to be transformed into the required format. The two main components of this layer are the *map manager* and the *repository manager*.

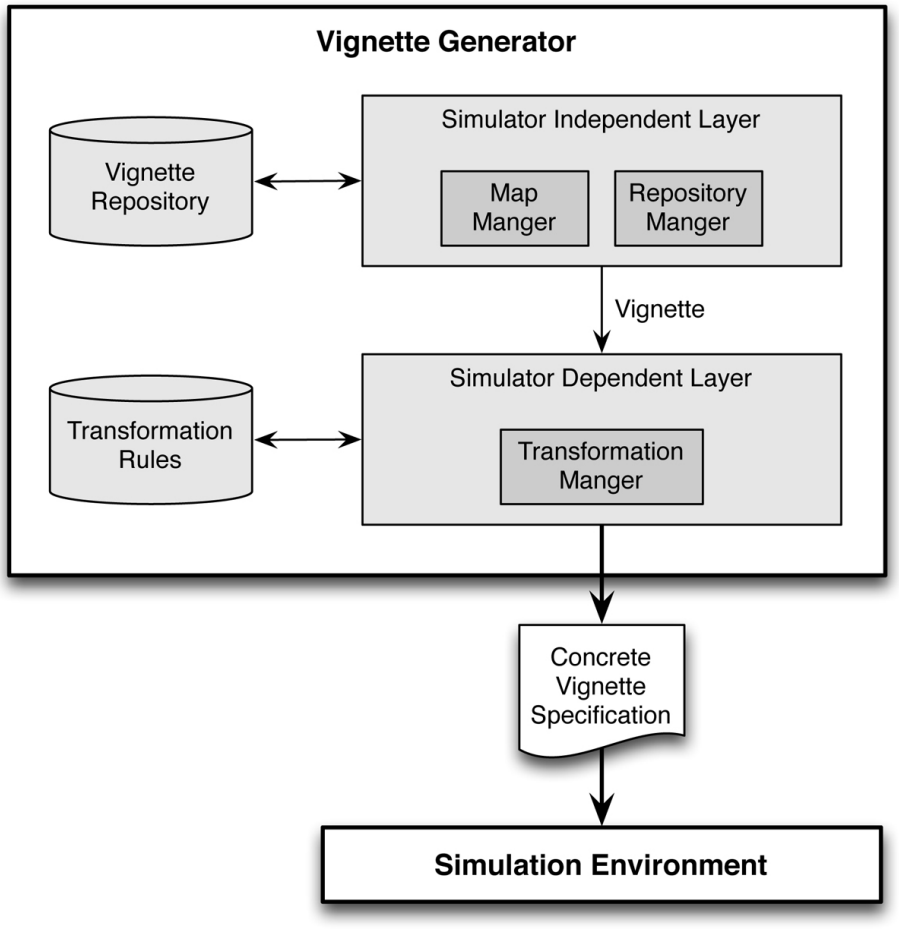

Figure 4 Vignette Generator Architecture.

- *Simulator Dependent Layer*: This layer transforms the output of the *simulator independent layer* into a format that can be processed and executed by the simulation environment. The main component of this layer is the *transformation manager*.
- *Vignette Repository*: The *vignette repository* is a knowledge base for storing all information about the $\mathcal{ET}$s, their attributes and relationships.
- *Transformation Rules*: Transformation rules which are used by the *simulator dependent layer* are stored here.

5 Formal Representation

Based on common concepts of computational logic and discrete mathematics, we formalize vignette specifications as *abstract state machine* (ASM) models [7,14]. Abstract state machines are known for their versatility in modeling semantic properties of algorithms, architectures, languages, protocols, and virtually all kinds of sequential, parallel and distributed systems. Building on an abstract machine framework has a number of advantages for defining structural and behavioral properties of vignettes, and the composition of simple and complex vignette elements so as to form complex structures.

5.1 Abstract State Machines and CoreASM Tool Environment

This section briefly presents the basic modeling concepts of Abstract State Machines and the CoreASM modeling framework for simulation and analysis of ASM specification.

5.1.1 Abstract State Machines

The ASM method [7] is a versatile mathematical framework for modeling virtually all kinds of discrete dynamic systems, including sequential, parallel and distributed systems, at any desired level of abstraction with a noticeable orientation toward practical applications. Building on common and widely used concepts from discrete mathematics and computational logic, it combines *abstract states* with *transition systems*. Abstract state machines are known for their semantic foundations for architectures, languages, and protocols, including some of the most prominent ones such as Java, SDL, and VHDL [15-20].

We use here an asynchronous ASM computation model, i.e., Distributed Abstract State Machine (DASM), with a non-empty set of autonomously operating computational *agents*. This set can change dynamically over machine runs so as to model a varying number of computational resources. The asynchronous computation model of DASM defines concurrent and reactive behavior, as observable in distributed computations performed by autonomously operating computational agents, according to the underlying semantic model. Depending on the agent type, agents have different dynamic properties as defined by their ASM program. Agents interact with each other and their operational environment by reading and writing shared locations of global machine *states*, represented as first-order structures [21] in terms of sets and operations defined thereon. Agents access and manipulate machine states as described by the transition rules that form their program. The underlying semantic model resolves potential conflicts according to the definition of *partially ordered runs* [7].

5.1.2 The CoreASM Extensible Architecture

CoreASM [14,22] is an Open Source project and tool suite for rapid proto-typing, analysis and experimental validation of ASM models. The tool suite provides a platform-independent execution engine and a GUI for interactive visualization and control of simulation runs. Building on an extensible plugin-based architecture [23], the CoreASM kernel (the *core* of the language and engine) contains only bare essentials; most of the language constructs and functionalities of the engine come in the form of plugins extending the kernel [24].

There are two flexible mechanisms for extending CoreASM. Plugins can either extend the functionality of specific engine components (such as the parser or the scheduler), introducing additional data or behavior to those components, or they can extend the control flow of the engine by interposing their own code in between state transitions of the engine. This extensibility offers a great deal of flexibility for customizing CoreASM depending on specific application needs.

Over several years, CoreASM has been put to the test in a range of R&D applications in commercial enterprises and government agencies, spanning computational criminology, coastal surveillance, SA, decision support systems, and Web services. CoreASM is implemented in Java under Academic Free License version 3.0 (AFL 3.0), providing a sensible compromise between public availability of the original source code and the existence of proprietary extensions for commercial applications. It is readily available at www.coreasm.org.

5.2 ASM Models of Vignettes

Intuitively, a vignette corresponds to an abstract machine *program*, and a vignette execution corresponds to an abstract machine *run*. Specifically, we use the asynchronous computation model of distributed abstract state machines, defined in terms of autonomously operating computational agents interacting with each other and their operational environment (the external world) by reading and writing locations of globally shared states. Potential conflicts are resolved by the underlying semantic model of partially ordered runs (see [7] for details).

Figure 5 illustrates a sample ASM model for a Rendezvousing Pattern between two agents at a high level of abstraction. The rendezvousing operation described here assumes that agent A_1 is a larger stationary vessel (for instance, a cargo ship), while A_2 is a smaller boat going back and forth between A_1 and a beach location to load and unload smuggled goods. The operation Rendezvousing(..) specifies the decomposition of a vignette into its constituent elements, expressing how A_1 and A_2 move to the offshore rendezvousing point and start the loading and unloading process. Exact coordinates for the rendezvous location and the beach point, as well as the detailed trajectory of A_2, are to be determined at runtime by the simulator executing the vignette.

A sample ASM model for a Rescuing Pattern among three agents is shown in Figure 6. In this SAR operation, a helicopter agent A_2 first searches and then identifies the location of the fishing boat agent A_1 in a distress situation. After locating the fishing boat, a coast guard vessel agent A_3 first extracts persons from the fishing boat and then secures the boat. The operation MoveAgent(..) is defined in the Rendezvousing example.

```
AGENT ≡ {CARGOSHIP, ZODIAC, ...}
MODE ≡ {APPROACHING, SMUGGLING, ...}
SMUGGLINGMODE ≡ {LOADING, UNLOADING, ...}
universe POSITION    // Geographic coordinates
universe OPERATION    // Operations of a vignette

AgentPosition : AGENT ↦ POSITION
Mode : OPERATION ↦ MODE
SmugglingMode : OPERATION ↦ SMUGGLINGMODE
BeachPoint : OPERATION ↦ POSITION
RendezvousPoint : OPERATION ↦ POSITION

Rendezvousing(A1, A2 : AGENT; Op : OPERATION) ≡
  if Mode(Op) = APPROACHING then
    MoveAgent(A1, RendezvousPoint(Op))
  if Mode(Op) = SMUGGLING then
    if SmugglingMode(Op) = LOADING then
      MoveAgent(A, RendezvousPoint(Op))
    if SmugglingMode(Op) = UNLOADING then
      MoveAgent(A, BeachPoint(Op))

MoveAgent(A : AGENT; P : POSITION) ≡
  NextPosition(A, P),   ...
  // Incrementally generates a trajectory by computing intermediate agents positions
  // depending on various factors, gradually moving an agent towards its destination.
```

Figure 5 ASM model of a Rendezvous Pattern between two agents A_1, A_2 in Territorial Sea (a belt of coastal waters extending from the baseline of a coastal state): A_1 is of type CARGOSHIP and A_2 is of type ZODIAC.

```
AGENT ≡ {FISHINGBOAT, HELICOPTER, COASTGUARDBOAT...}
MODE ≡ {SEARCHING, RESCUING, SECURING, ...}
universe POSITION    // Geographic coordinates
universe AREA    // Geographic areas
universe OPERATION    // Operations of a vignette

AgentPosition : AGENT ↦ POSITION
Mode : OPERATION ↦ MODE
Area : OPERATION ↦ AREA

Rescuing(A1, A2, A3 : AGENT; Op : OPERATION) ≡
  if AgentPosition(A1) = undef then
    if Mode(Op) = SEARCHING then
      SearchArea(A1, A2, Area(Op))
      // Searches for A1 in a specified area and sets the position of A1 once identified.
  if AgentPosition(A1) ≠ undef then
    if Mode(Op) = RESCUING then
      RescuePersons(A1, A3)
    else if Mode(Op) = SECURING then
      SecureBoat(A1, A3)

RescuePersons(A1, A3 : AGENT) ≡
  if AgentPosition(A1) ≠ AgentPosition(A3) then
    MoveAgent(A3, AgentPosition(A1))
  else
    PerformRescue(A1, A3)

SecureBoat(A1, A3 : AGENT) ≡
  if AgentPosition(A1) ≠ AgentPosition(A3) then
    MoveAgent(A3, AgentPosition(A1))
  else
    PerformSecure(A1, A3)
```

Figure 6 ASM model of Rescuing Pattern among three agents in Territorial Sea: Agent A_1 is of type FISHINGBOAT, A_2 is of type HELICOPTER, and A_3 is of type COASTGUARDBOAT.

6 Applications

An operational prototype of the Vignette Generator has been designed and developed in Java based on the concepts presented in this paper. It is capable of generating a wide spectrum of vignette specifications for the INFORM Lab simulation environment (see Section 6.1.2). A screenshot of the Vignette Generator illustrating a complex SAR scenario is illustrated in Figure 7.

The Vignette Generator has the potential to interoperate with different simulation environments, namely those for which the corresponding interface configuration format is determined, due to the following features:

- *Composable Vignette Element Type*: Vignette $\mathcal{V}$ is generated from $\mathcal{ET}$s forming the basic building blocks. So, new $\mathcal{ET}$s can be generated based on existing ones at run-time, which means that $\mathcal{R}$ is *flexible* enough that users can easily *extend* the set of $\mathcal{ET}$s, and consequently $\mathcal{R}$.
- *Separated Transformation Layer at the Architecture Level*: Generating a vignette $\mathcal{V}$ from high-level abstract $\mathcal{ET}$s, and transforming $\mathcal{V}$ into an executable format (for the simulation environment) are separated into two different architectural layers. The input format of the simulation environment is irrelevant for the composition of $\mathcal{ET}$s and $\mathcal{E}$ s. This way, the two-layer architecture enhances of the *portability* of the Vignette Generator.

Therefore, we are able to generate various vignettes by instantiating $\mathcal{ET}$s with less consideration of the input format required by a specific simulation environment. We have designed the Vignette Generator such that it can provide different simulation environments with appropriate inputs without significant modifications. In the following section, realistic applications of the tool are discussed.

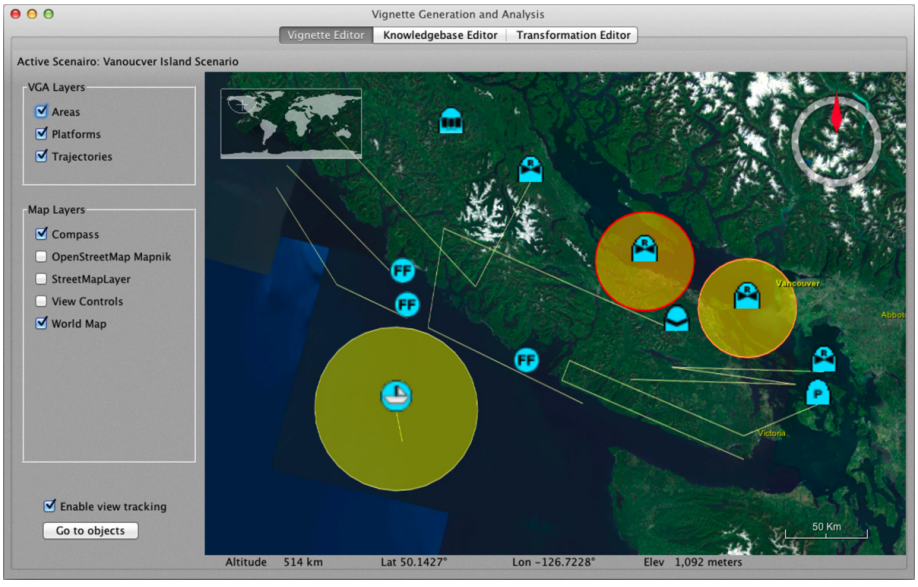

Figure 7 A Screenshot of a Map-Based Scenario Representation in the Vignette Generator.

6.1 Generating Application Domain Vignettes

A detailed sample scenario, including a number of related vignettes, serves to illustrate the functionality of the proposed Vignette Generator. The generated vignettes, which can be executed in the INFORM Lab simulation environment, are based on the sample scenario presented in Section 6.1.1. In this scenario two different kinds of vignettes are presented: cooperative search, such as locating a fishing boat in distress, and non-cooperative search to detect and prevent illegal activities (for instance, smuggling operations).

6.1.1 Sample Scenario and Vignettes

Consider a simulation of maritime activities near Vancouver Island, B.C., Canada on December 26, 2011 between 6:30 AM and 7:30 AM. These maritime activities include the following events and stories:

- December 26, 2011 - 6:30 AM:
  - Routine patrolling of Vancouver Island's coastal region by *coast guard vessels* and *helicopters*.
  - Monitoring the maritime environment through *satellite* images.
- December 26, 2011 - 6:45 AM:
  - Receiving a rescue request from a *fishing boat* in a distress situation with an approximate reported location southeast of the Gulf Islands in the Strait of Georgia between Vancouver Island and the mainland of British Columbia.
- December 26, 2011 - 6:50 AM:
  - Observing suspicious offshore activities between a *cargo ship* and one or more *small boats (zodiacs)* roughly 50 kilometres northwest of Port Hardy at the northern end of Vancouver Island.
- December 26, 2011 - 7:00 AM:
  - Searching and rescuing the passengers of the *fishing boat* and securing their boat as a cooperative SAR mission.
- December 26, 2011 - 7:10 AM:
  - Analyzing the suspicious offshore activities by a *coast guard helicopter* confirms the suspicion of a smuggling operation in progress.
- In addition to these vignettes, there are also a number of *fishing boats*, *sailing boats*, *cargo vessels*, *passenger vessels*, and *tankers* operating in the area of interest. Since one of the main purposes of the Vignette Generator is to test different situation analysis methods and information fusion algorithms, considering such "background traffic" is necessary to produce realistic noise, say for testing offshore smuggling operation detection algorithms under realistic conditions.

The following four vignettes are elicited from these stories:

1. Routine patrolling and monitoring of the coastal regions of Vancouver Island by *coast guard vessels*, *helicopters*, and *satellites* (see Figure 8(a)).

2. Searching and rescuing a *fishing boat* in a distress situation by *coast guard vessels* and *SAR helicopters* (see Figure 8(b)).

3. Observing and detecting suspicious activities between a *cargo ship* and *two zodiacs* (see Figure 8(c)).

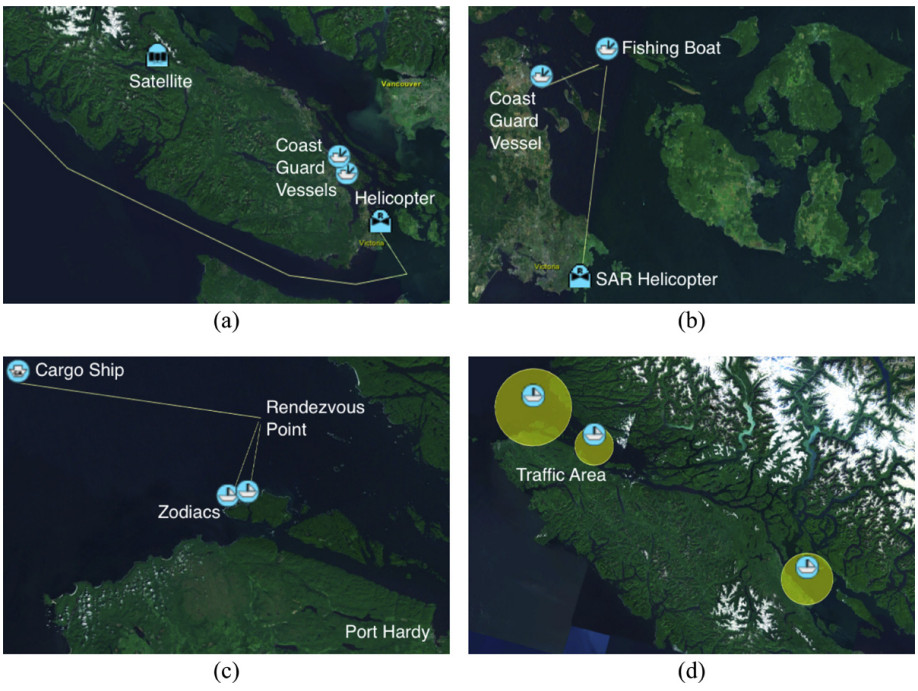

Figure 8 Sample Vignettes from the Vignette Generator (a) Routine Patrolling: Coast guard vessels, helicopters, and satellite images are used to patrol and monitor the area of interest; (b) Search and Rescue: A fishing boat in a distress situation is located southeast of the Gulf Islands in the Strait of Georgia, and a coast guard vessel and a helicopter are performing SAR operation; (c) Rendezvousing: Suspicious offshore activities between a cargo ship and two zodiacs are observed roughly 50 kilometres northwest of Port Hardy; (d) Traffic Area: There are a number different kinds of vessels (including fishing boats, sailing boats, passenger vessels, cargo vessels, and tankers) operating in the area of interest.

4. Random movements of various types of smaller boats and larger vessels in a maritime environment (see Figure 8(d)).

Combinations of the above high-level vignettes (instantiated with a number of non-deterministically chosen parameter values) can be defined in the Vignette Generator. As a result, one can easily generate a number of concrete instances of these vignettes as input for the INFORM Lab simulation environment. Figure 9 presents a screenshot from the INFORM Lab simulation of a concrete instance of the vignette shown in Figure 7.

The examples described here are well founded in real-world situations and have been examined thoroughly and validated by domain experts. However, it is important to note that they describe a situation at a very local scale. More complex vignettes can involve hundreds of agents and platforms, as well as numerous distinct behaviors. The Vignette Generator is fully capable of handling vignettes of high complexity and large scale, but a full description of such a comprehensive example would be outside of the scope of a journal article.

6.1.2 INFORM Lab Simulation Environment

INFORM Lab, designed and developed at MDA Corporation, Richmond, B.C., Canada, is an advanced agent based simulation framework built around the OODA paradigm (Observe, Orient, Decide, Act) [8]. It is used to simulate Coastal Wide Area Surveillance applications, including SAR and detection of illegal activities (see Figure 9).

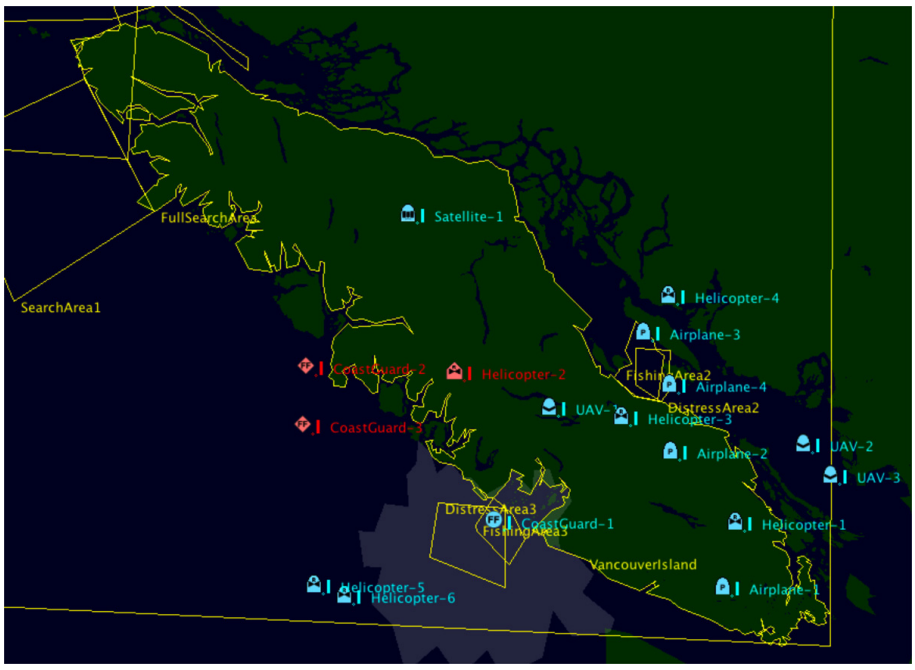

Figure 9 A Screenshot of the INFORM Lab's Simulator: Detection of a Smuggling Operation in the Strait of Georgia at the northern peak of Vancouver Island.

It allows experimentation with higher-level distributed dynamic information fusion, distributed dynamic resource management, communication strategies, and configuration management given multiple constraints on resources and their communications networks. So, it can be considered as a testbed which also allows the evaluation of sensing strategies, motion strategies, and a range of control strategies from independent agent operation through various levels of agent collaboration.

The INFORM Lab simulator is driven by a configuration specification file that specifies the agents and their relationships, the environment, initial conditions as well as any scripted events. Setting up this file can be time-consuming, difficult and error prone. However, being a testbed, INFORM Lab requires that statistically relevant metrics can be extracted from the results. This means that many small variations of an experiment need to be run. Specifying this in a convenient, systematic, and error-free manner would be a huge task without a vignette generator. The Vignette Generator can provide the required variability of the experiments with great convenience and is time-saving for the experimenter.

6.2 Anomaly Detection

Related to the work presented here is a research project on a model-driven framework for engineering Situation Analysis Decision Support (SADS) systems for the domain of marine safety & security [25,26]. SADS system engineering practices call for systematic formal modeling approaches to manage complexity through modularization, refinement and validation of abstract models. In this light, we explore SADS system design based on ASM modeling techniques paired with CoreASM tool support to facilitate analysis of the problem space and reasoning about design decisions and conformance criteria so as to ensure they are properly established and well understood prior to

building the system. We provide an extension to CoreASM for the marine safety & security domain, Specifically for capturing rendezvous scenarios. The extension yields the necessary background concepts, such as mobile sensors and shipping lanes, and offers runtime visualization of simulation runs together with an analyzer to measure success of various rendezvous detection strategies used in the model. The proposed framework complements purely analytical means that focus on verification of internal properties, such as consistency and completeness of a model, and provides a sensible way of linking formal and empirical aspects in the model-driven engineering of SADS systems. Experimental studies of SA scenarios can considerably enhance our insight into intricate system dynamics and simplify the challenging task of deriving meaningful conformance criteria for checking the validity of SADS domain models against established operational concepts of marine safety & security. We illustrate the application of the proposed approach using sample rendezvous scenarios (see [25] for a detailed scenario description and results). Work in progress is now extending the scope of our SADS model to capture a much wider range of observable behaviors of marine traffic that deviate from what is considered normal. This work aims at integrating model-driven approaches with data-driven approaches of behavior detection into a hybrid detection framework operating on real-world data sets.

6.3 General Benefits of Using the Vignette Generator

- *Ease of Use*: Graphical user interface facilitates manipulation of the repository and the transformation rules, and helps users to generate appropriate test cases.
- *Reusability*: By gradually building a repository of $\mathcal{ET}$s, we are able to reuse the defined $\mathcal{ET}$s in future applications.
- *Compositionality*: By using the proposed composition mechanisms, we are able to generate increasingly more complex $\mathcal{ET}$s and $\mathcal{V}$ s by combining and orchestrating existing $\mathcal{ET}$s.
- *Complexity Management*: The hierarchical structure of the repository reduces the complexity of generating vignettes. Low-level details for an $\mathcal{ET}$ need only be defined once and can then be reused. Further, a composite $\mathcal{ET}$ hides the low-level details of its internal elements without preventing access to them.
- *Incremental Improvement*: Over time, generating complex vignettes becomes increasingly efficient as the repository grows, offering more and richer choices.

7 Conclusion and Future Work

This paper addresses a notorious problem in testing computational models for infrastructure protection and emergency response by means of simulation and animation using realistic scenarios. Simulation plays a key role for analyzing decision support systems, situation analysis methods, and information fusion algorithms. Considering their inherent complexity, the validity of realistic models need to be established through progressive comprehension of the real-world phenomena being studied. This can be done by gradually improving the applied methods in an interactive manner. However, such improvements are virtually impossible without preforming in-depth experiments. Furthermore, manual generation of meaningful vignettes is tedious and cumbersome,

and generating such vignettes even in an automatic or semi-automatic fashion is a complex and challenging engineering task.

We have presented the conceptual design for a vignette generator to overcome this problem, and described its high-level requirements and architecture in detail. Further, we have illustrated a formal approach to vignette specification in terms of a sample vignette, building on well-defined composition mechanisms. As justified in the paper, the proposed design ensures basic quality attributes, including *flexibility*, *extensibility*, *reusability*, *scalability*, and *portability*. Practical experience has been gained from using the Vignette Generator with the INFORM Lab simulation environment at MDA Corporation for developing situation analysis decision support models. Generating vignettes for marine safety & security scenarios undeniably helps developers to test and evaluate their algorithms. It also helps end-users to simulate and analyze real-world situations. Arguably, the scope of potential applications of the Vignette Generator presented here extends far beyond the domain of marine safety & security operations as many central concepts directly carry over to a much broader range of infrastructure protection and emergency response scenarios.

Acknowledgements
The work presented here has been funded by MDA Corporation, MITACS, and NSERC under NSERC Collaborative Research and Development Grant No. 342503-06.

Author details
[1]Software Technology Lab, Computing Science, Simon Fraser University, B.C., Canada [2]SAP Research, Karlsruhe, Germany [3]MDA Corporation, Research & Development, Richmond, B.C., Canada

Authors' contributions
This paper is the outcome of a joint effort with major contributions particularly from the first three authors. All authors have contributed to the paper, read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Received: 9 September 2011 Accepted: 27 February 2012 Published: 27 February 2012

References

1. Natural Resources Canada: **Natural Resources Canada. The Atlas of Canada - Coastline and Shoreline [Online].** 2011 [http://atlas.nrcan.gc.ca/site/english/learningresources/facts/coastline.html], [Last visited, August 2011].
2. Bossé É, Roy J, Ward S: *Models and Tools for Information Fusion* Artech House; 2007.
3. Endsley MR: **Theoretical Underpinnings of Situation Awareness: A Critical Review.** In *Situation Awareness Analysis and Measurement.* Edited by: Endsley MR, Garland DJ. LEA; 2000:3-32.
4. Glässer U, Jackson P, Araghi AK, Shahir HY, Wehn H: **A Collaborative Decision Support Model for Marine Safety and Security Operations.** In *Distributed, Parallel and Biologically Inspired Systems.* Edited by: Hinchey M, et al. Springer; 2010:.
5. Glässer U, Jackson P, Araghi AK, Shahir HY: **Intelligent Decision Support for Marine Safety and Security Operations.** In *Proceedings of the 2010 IEEE International Conference on Intelligence and Security Informatics, Vancouver, Canada.* Edited by: Yang C, et al. IEEE; 2010:101-107.
6. Lambert D, Wark S, Bossé É, Pigeon L, Blackman C, Hinma M: **A Coalition Approach to Higher-Level Fusion.** *Proceedings of the International Conference on Information Fusion, Seattle, USA* IEEE; 2009.
7. Börger E, Stärk R: *Abstract State Machines: A Method for High-Level System Design and Analysis* Springer-Verlag; 2003.
8. Li Z, Leung H, Valin P, Wehn H: **High Level data fusion system for CanCoastWatch.** *Proceedings of the 10th International Conference on Information Fusion* 2007, 1-6.
9. Jackson P, Glässer U, Shahir HY, Wehn H: **An Extensible Decision Engine for Marine Safety and Security.** *Proceedings of the 2011 IEEE International Conference on Intelligence and Security Informatics, Beijing, China* IEEE; 2011, 54-59.
10. Shahir HY, Glässer U, Jackson P, Wehn H: **Test-Case Generation for Marine Safety and Security Scenarios.** *Proceedings of the 2011 IEEE International Conference on Intelligence and Security Informatics, Beijing, China* IEEE; 2011, 48-53.
11. Royal Canadian Mounted Police: **Marine Security Operation Centres (MSOC) [Online].** 2011 [http://www.grc-rcmp.gc.ca/mari-port/msoc-cosm-eng.htm], [Last visited, August 2011].
12. MarineTraffic Service: **Vessel positions tracking based on AIS data.** 2011 [http://www.marinetraffic.com/], [Last visited, August 2011].
13. U.S. Department of Homeland Security - U.S. Coast Guard: **Automatic Identification System Overview.** 2011 [http://www.navcen.uscg.gov/?pageName=AIS], [Last visited, August 2011].

14. Farahbod R, Gervasi V, Glässer U: **CoreASM: An Extensible ASM Execution Engine.** *Fundamenta Informaticae* 2007, **77(1-2)**:71-103.
15. Glässer U, Gotzhein R, Prinz A: **The Formal Semantics of SDL-2000: Status and Perspectives.** *Computer Networks* 2003, **42(3)**:343-358.
16. Börger E, Glässer U, Müller W: **Formal Definition of an Abstract VHDL'93 Simulator by EA-Machines.** In *Formal Semantics for VHDL*. Edited by: Delgado Kloos C, Breuer PT. Kluwer Academic Publishers; 1995:107-139.
17. Glässer U, Gurevich Y, Veanes M: **Abstract Communication Model for Distributed Systems.** *IEEE Trans on Soft Eng* 2004, **30(7)**:458-472.
18. Stärk R, Schmid J, Börger E: *Java and the Java Virtual Machine: Definition, Verication, Validation* Springer-Verlag; 2001.
19. Farahbod R, Glässer U, Vajihollahi M: **An Abstract Machine Architecture for Web Service Based Business Process Management.** *International Journal of Business Process Integration and Management* 2007, **1**:279-291.
20. Börger E, Riccobene E, Schmid J: **Capturing Requirements by Abstract State Machines: The Light Control Case Study.** *Journal of Universal Computer Science* 2000, **6(7)**:597-620.
21. Enderton HB: *A Mathematical Introduction to Logic.* second edition. Harcourt Academic Press; 2001.
22. Farahbod R, Glässer U: **The CoreASM Modeling Framework.** *Software: Practice and Experience* 2011, **41(2)**:167-178.
23. Farahbod R, Gervasi V, Glässer U, Ma G: **CoreASM Plug-in Architecture.** In *Rigorous Methods for Software Construction and Analysis.* Edited by: Abrial JR, Glässer U. Springer; 2009:147-169, LNCS Festschrift volume 5115.
24. Farahbod R: **CoreASM: An Extensible Modeling Framework & Tool Environment for High-level Design and Analysis of Distributed Systems.** *PhD thesis* Simon Fraser University, BC, Canada; 2009.
25. Farahbod R, Avram V, Glässer U, Guitouni A: **Engineering Situation Analysis Decision Support Systems.** *Proceedings of the European Intelligence and Security Informatics Conference, Athens, Greece* 2011, 10-18.
26. Farahbod R, Avram V, Glässer U, Guitouni A: **A Formal Engineering Approach to High-level Design of Situation Analysis Decision Support Systems.** *Proceedings of the 13th International Conference on Formal Engineering Methods, Durham, UK* 2011, 211-226.

doi:10.1186/2190-8532-1-4
Cite this article as: Shahir *et al.*: Generating test cases for marine safety and security scenarios: a composition framework. *Security Informatics* 2012 **1**:4.