



Driving Innovation in Crisis Management  
for European Resilience



## D923.21 FIRST RELEASE OF THE TEST-BED REFERENCE IMPLEMENTATION

SP92 - TEST-BED

MARCH 2018 (M47)



This project has received funding from the European Union's 7th Framework Programme for Research, Technological Development and Demonstration under Grant Agreement (GA) N° #607798

---

# Table of Contents

About

Executive summary

1. Introduction

2. Test-bed description

3. Test-bed for Trial owners

4. Test-bed for developers & sysops

5. Test-bed design

Acronyms

Annex 1

List of Figures

---

## Project information

<b>Project Acronym:</b>	DRIVER+
<b>Project Full Title:</b>	Driving Innovation in Crisis Management for European Resilience
<b>Grant Agreement:</b>	607798
<b>Project Duration:</b>	72 months (May 2014 - April 2020)
<b>Project Technical Coordinator:</b>	TNO
<b>Contact:</b>	coordination@projectdriver.eu

## Document information

<b>Document Status:</b>	Final
<b>Document Title:</b>	D923.21 Test-bed reference implementation
<b>Document Nature:</b>	Report (R)
<b>Dissemination Level:</b>	Public (P)
<b>Due Date:</b>	31 March, 2018
<b>Submission Date:</b>	03/04/2018
<b>Sub-Project (SP):</b>	SP92 - Test-bed
<b>Work Package (WP):</b>	WP923 – Test-bed infrastructure
<b>Document Leader:</b>	TNO
<b>File Name:</b>	test-bed-design.pdf, latest version available online at <a href="https://driver-eu.gitbooks.io/test-bed-specification">driver-eu.gitbooks.io/test-bed-specification</a>

**DISCLAIMER**

*The opinion stated in this report reflects the opinion of the authors and not the opinion of the European Commission. All intellectual property rights are owned by the DRIVER+ consortium members and are protected by the applicable laws. Except where otherwise specified, all document contents are: “©DRIVER+ Project - All rights reserved”.*

*Reproduction is not authorised without prior written agreement.*

*The commercial use of any information contained in this document may require a license from the owner of that information.*

*All DRIVER+ consortium members are also committed to publish accurate and up to date information and take the greatest care to do so. However, the DRIVER+ consortium members cannot accept liability for any inaccuracies or omissions nor do they accept liability for any direct, indirect, special, consequential or other losses or damages of any kind arising out of the use of this information.*

## Revision table

Issue	Date	Comment	Author
V0.1	01/02/2018	Initial draft	Erik Vullings (TNO, WP923 leader)
V0.2	21/03/2018	Review process started	Erik Vullings
V0.3	26/03/2018	Reviews from Alessandro Annunziato (JRC) and Héctor Naranjo (GMV) processed	Erik Vullings
V0.4	28/03/2018	Reviews from Chiara Fonio (JRC), Laurent Dubois (Thales) and Peter Petiet (TNO) processed	Erik Vullings
V0.5	29/03/2018	Reviews from Marcel van Berlo (TNO) processed	Erik Vullings
V0.6	03/04/2018	Final check and approval for submission	Peter Petiet, Project Director (TNO)
V1.0	03/04/2018	Submission to the EC	Francisco Gala (ATOS)

# The DRIVER+ project

Current and future challenges due to increasingly severe consequences of natural disasters and terrorist threats require the development and uptake of innovative solutions that are addressing the operational needs of practitioners dealing with Crisis Management.

DRIVER+ (Driving Innovation in Crisis Management for European Resilience) is a FP7 Crisis Management demonstration project aiming at improving the way capability development and innovation management is tackled. DRIVER+ has three main objectives:

1. Develop a pan-European Test-bed for Crisis Management capability development:
  - Develop a common guidance methodology and tool (supporting Trials and the gathering of lessons learned).
  - Develop an infrastructure to create relevant environments, for enabling the trialling of new solutions and to explore and share Crisis Management capabilities.
  - Run Trials in order to assess the value of solutions addressing specific needs using guidance and infrastructure.
  - Ensure the sustainability of the pan-European Test-bed.
2. Develop a well-balanced comprehensive Portfolio of Crisis Management Solutions:
  - Facilitate the usage of the Portfolio of Solutions.
  - Ensure the sustainability of the Portfolio of Solutions.
3. Facilitate a shared understanding of Crisis Management across Europe:
  - Establish a common background.
  - Cooperate with external partners in joint Trials.
  - Disseminate project results.

To achieve these objectives, five Subprojects (SPs) have been established. **SP91 Project Management** is devoted to consortium level project management, and it is also in charge of the alignment of DRIVER+ with external initiatives on crisis management for the benefit of DRIVER+ and its stakeholders. In DRIVER+, all activities related to societal impact assessment (from the former SP8 and SP9) are part of SP91 as well. **SP92 Test-bed** will deliver a guidance methodology and guidance tool supporting the design, conduct and analysis of Trials and will develop a reference implementation of the Test-bed. It will also create the scenario simulation capability to support execution of the Trials. **SP93 Solutions** will deliver the Portfolio of Solutions which is a database driven web site that documents all the available DRIVER+ solutions, as well as solutions from external organisations. Adapting solutions to fit the needs addressed in Trials will be done in SP93. **SP94 Trials** will organize four series of Trials as well as the final demo. **SP95 Impact, Engagement and Sustainability**, oversees communication and dissemination, and also addresses issues related to improving sustainability, market aspects of solutions, and standardization.

The DRIVER+ Trials and the Final Demonstration will benefit from the DRIVER+ Test-bed, providing the technological infrastructure, the necessary supporting methodology and adequate support tools to prepare, conduct and evaluate the Trials. All results from the Trials will be stored and made available in the Portfolio of Solutions, being a central platform to present innovative solutions from consortium partners and third parties and to share experiences and best practices with respect to their application. In order to enhance the current European cooperation framework within the Crisis Management domain and to facilitate a shared understanding of Crisis Management across Europe, DRIVER+ will carry out a wide range of activities, whose most important will be to build and structure a dedicated community of practice in crisis management, thereby connecting and fostering the exchange on lessons learnt and best practices between Crisis Management practitioners as well as technological solution providers.

# Executive Summary

The Test-bed reference implementation lies at the heart of the Trialling environment of the [DRIVER+ project](#) (see Figure 1). It provides an open source technical backbone to perform Trials or exercises in a methodical and structured way by offering practitioners a suite of free *software* tools. This document discusses the Test-bed's usage and design, and therefore it is expected that the reader has at least some technical background. In DRIVER+ deliverable [D923.11](#) "Functional specification of the Test-bed", the requirements of the Test-bed are documented. These requirements serve as basis for the design of the first release of the DRIVER+ Test-bed reference implementation as described in this deliverable D923.21.

The [trial](#)-oriented environment developed in sub-project 92 (Test-bed) of DRIVER+ is conceived and designed to allow systematic testing of solutions in realistic but non-operational contexts (namely, in Trials) to help practitioners in assessing solutions that can drive innovation (changes) before adopting them. See also deliverable D922.21, "[Trial](#) guidance methodology and guidance tool specifications (version 1)".

The purpose of conducting Trials in DRIVER+ is to find out if and how some innovative solutions can help resolve the needs of Crisis Management practitioners.

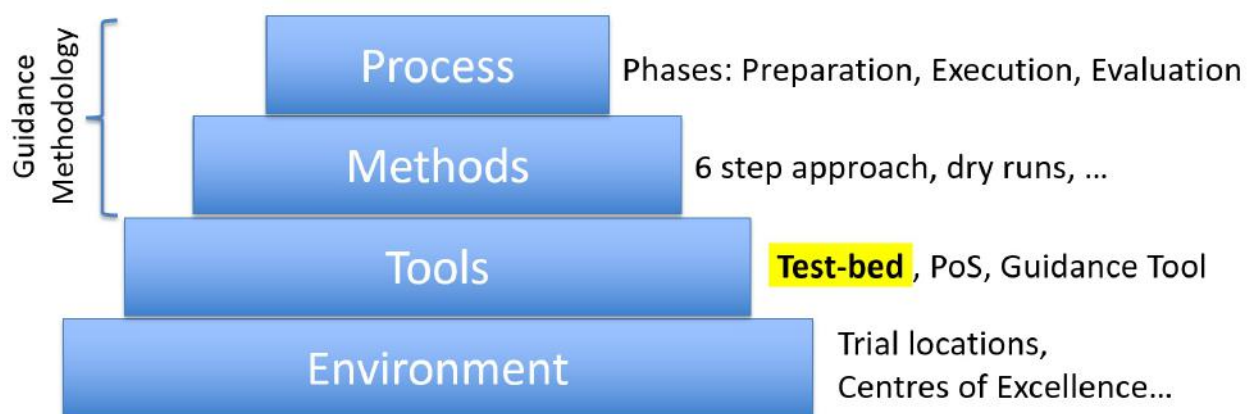


Figure 1. PTME paradigm applied to DRIVER+.

The basic problem that the Test-bed reference implementation tries to solve is how to connect different solutions, which solve a particular Crisis Management ([CM](#)) gap, to:

- **Each other**: since no single application can solve all [CM](#) gaps, they need to work together by sharing information.
- **One or more simulators**: since during a [Trial](#), you cannot start a real incident, there needs to be a way to *simulate* a realistic incident.

The main design decision is to connect solutions to a so-called Common Information Space (CIS), simulators to a Common Simulation Space (CSS), and to have gateway services in between that selectively allow some information to pass between the two spaces (see Figure 2). These spaces can be public or private, and allow for applications to write messages to a topic of interest, or read those messages. Although both spaces are comparable in that they share well structured messages (using Apache AVRO) over a popular open source distributed messaging system, Apache Kafka, the separation allows for a better control of the message flow, and for replacing parts with an alternative implementations if needed.

In addition, several adapters are created to connect solutions and simulators to the CIS and CSS: besides allowing users of the Test-bed to choose an adapter in their favourite programming language, and share messages, it also provides a common interface for configuration, heartbeat messages, and security. Currently, adapters in Java, C#, JavaScript/TypeScript and REST are available, and a Python adapter is planned for the next release.

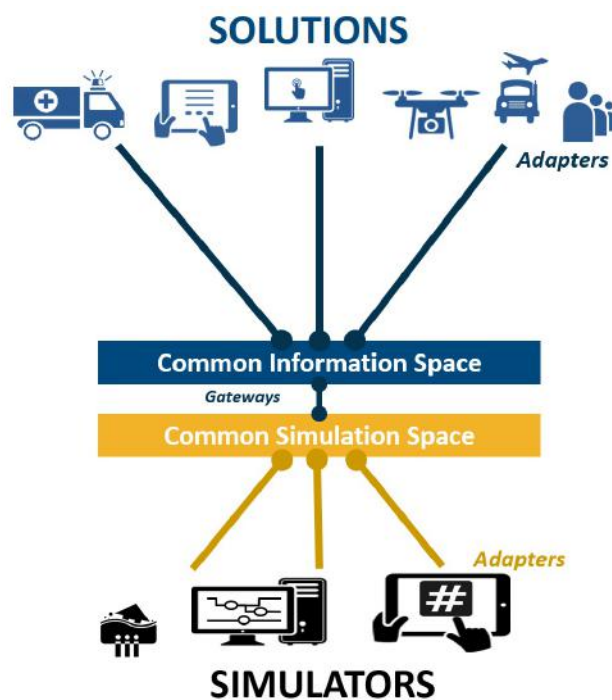


Figure 2. CIS and CSS.

Around this core functionality, additional tools are developed that facilitate the usage of this environment (see Figure 3):

- Administrative tools: Is everyone up-and-running, secure, and connected to the right information topics?
- Evaluation tools: What did we observe during the Trial, and what implications does this have during the After-Action Review.



- Scenario tools: To create an interesting scenario that triggers the participants and solutions in the right way.
- Support tools: For testing and debugging, for creating your personalized Test-bed environment, but also to share common data such as map layers or census data.

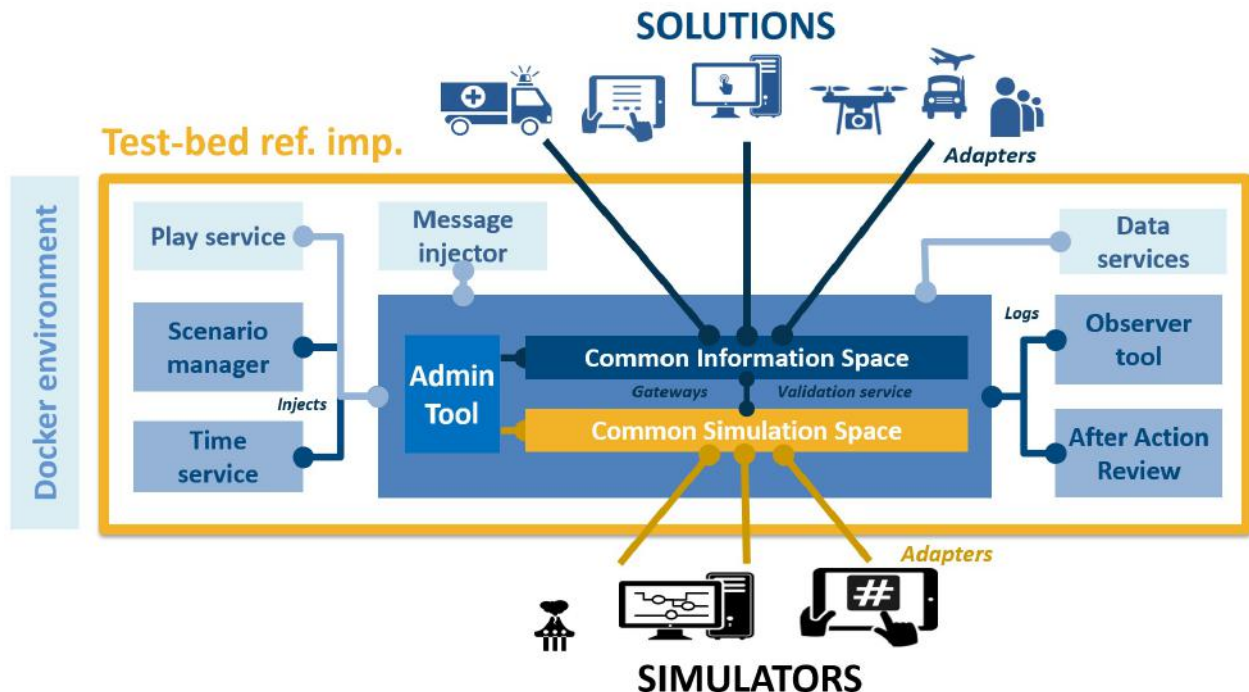


Figure 3. Test-bed reference implementation.

The Test-bed will be delivered in 3 versions. Version 1 is intended for use in Trials 1 and 2, and consists of a limited number of components (i.e. [Scenario Manager](#) and [AAR](#) not included and other components in first prototype quality). Version 2 is to be used in Trials 3 and 4 and the Final Demo and comes with all components with a quality level surpassing that of a first prototype. It should contain data-sets and basic scenarios that can be used for effectively implementing and testing the Test-bed. The final version, Version 3, should have an even better quality based on experiences gathered from the use of the Test-bed in the executed Trials. All versions are already available open-source (under a liberal MIT license) on [github.com/DRIVER-EU](https://github.com/DRIVER-EU). The subsequent updated Test-bed reference implementations are documented in D923.21, D923.22 and D923.23 - Reference Implementations of the Test-bed (versions 1/2/3 resp.).

This document is a living document, and the latest version is available [online at \[www.gitbook.com/book/driver-eu/test-bed-design\]\(https://www.gitbook.com/book/driver-eu/test-bed-design\)](https://www.gitbook.com/book/driver-eu/test-bed-design) or can be downloaded as PDF or ebook (epub or mobi) <sup>1</sup>.

<sup>1</sup>. Since the printed version is automatically generated from the online book, the template is slightly different from a Word-based version. It starts at a high-level with a description of the main components of the Test-bed reference implementation. Next, chapters are dedicated to the main users of this environment, practitioners and developers/ICT administrators. And it concludes with a brief explanation of the main design decisions. ↩

# 1. Introduction

In the Crisis Management (CM) domain, practitioners need to be prepared for the unexpected: based on past experience and the local incidents they had to deal with, they develop a feeling for the things that did not go so well. As with most incidents many lives are involved, they are continuously looking for solutions to improve their response and preparedness.

Within the DRIVER+ project, a Trial Guidance Methodology (TGM, deliverable D922.21 - Trial guidance methodology and guidance tool specifications (version 1)) and tools are developed to help resolve the needs of practitioners through a systematic and pragmatic approach. The Test-bed infrastructure is a suite of software tools and services that enables solutions and simulators in the Crisis Management (CM) domain to easily exchange information (see Figure 4). This allows end-users in the CM domain to Trial solutions, and see if they address their gaps. Additionally, it can be used to support training exercises as well.

The simplified DRIVER+ TGM process to Trial solutions is like this:

## 1. PREPARATION PHASE:

- The operational issues practitioners experience are matched with one or more of the well-known crisis management gaps, as experienced by many of their colleagues, e.g. *How to get a real-time and dynamic overview of the position of all personnel?*.
- These gaps, in turn, are still too generic to address, and are made more specific, leading to so-called 'research questions', e.g. *Which Situational Awareness-increasing solution fits best with our mode of operation?*
- Existing solutions are reviewed and selected. Some solutions can even be tried out standalone using the Test-bed and an existing mini-scenario.
- Based on the selected solutions, gaps and research questions, a data collection plan is developed: what kind of data does the Trial need to generate to enable a valid evaluation at the end of the Trial.
- A scenario is developed that puts these solutions to the test. This includes selecting and connecting simulators of a fictive incident, e.g. a flooding simulator, and other simulators to create a realistic environment, e.g. a traffic simulator.
- Selected solutions and existing legacy systems are connected to the Test-bed, so they can receive input from the simulators as well as each other.

## 2. EXECUTION PHASE:

The Test-bed is setup, all simulators and solutions are connected, data collection is in place, and a scenario can be run (executed). Before the actual **Trial**, several dry runs are performed, partially for testing the setup, and partially for training the participants in using the solutions. This phase ends when the **Trial** is executed and observed.

### 3. EVALUATION PHASE:

Based on the recorded observations and collected data (screenshots of running applications, messages sent, actions and decisions taken) during the **Trial**, the solutions are evaluated with the participants, leading to a good appreciation of how the selected solutions have contributed to solving the problems.

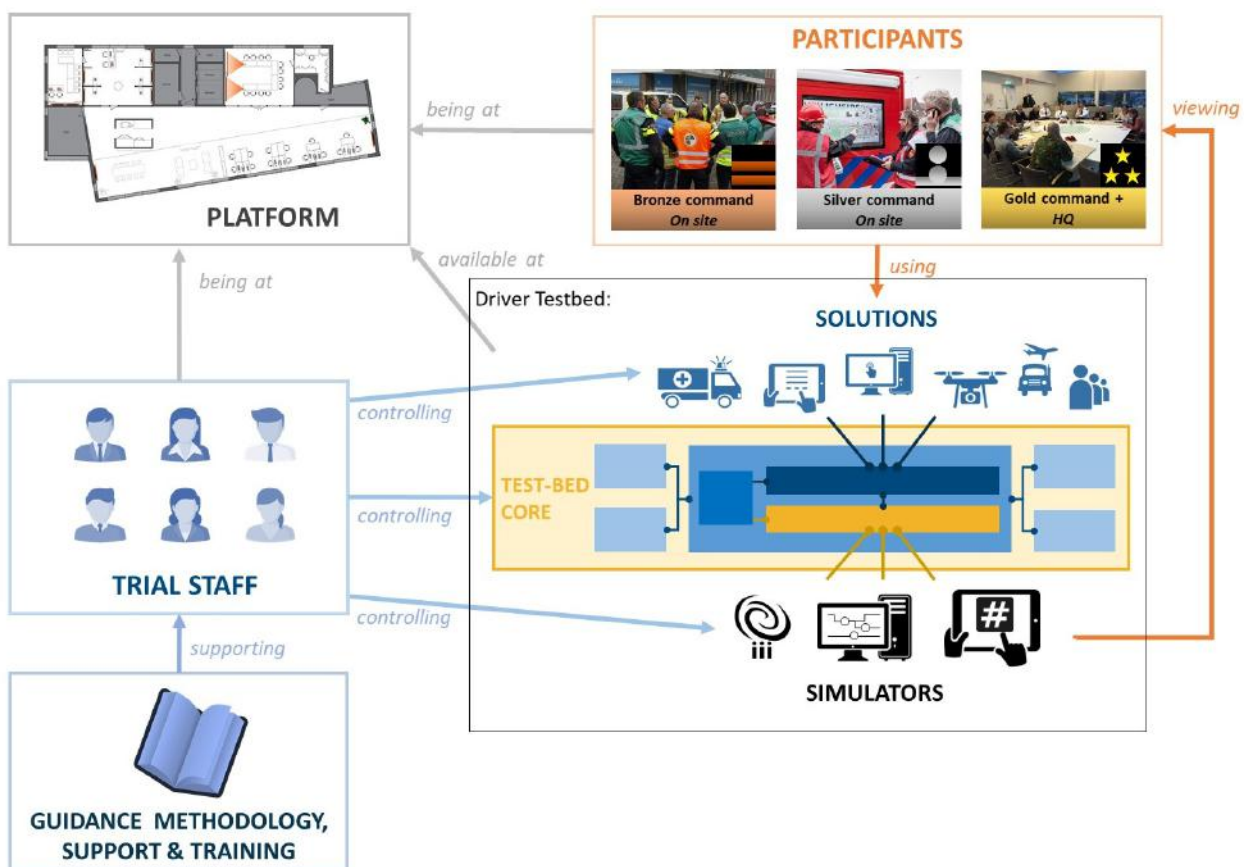


Figure 4. Test-bed environment.

## 1.1 Aim

The Test-bed supports practitioners by providing an environment in which they can easily **Trial** new solutions and run exercises. This has several implications for the test-bed:

- **Evaluation support:** As we are testing and evaluating new solutions, the Test-bed provides tools for observers and After-Action Review.

- **Simulation support:** Due to the nature of the crisis management domain, most solutions cannot be properly tested outside an actual crisis situation. As starting a flooding or burning a forest is clearly not an option to test the solutions, the crisis incident needs to be faked or simulated. The same applies to expected reactions of the environment: people panicking, traffic jams, etc. must be simulated too.
- **Execution support:** As the solutions are typically tested within an incident scenario, the Test-bed provides tools to create and execute scenario's.
- **Development support:** To connect new solutions and simulators to the Test-bed, the Test-bed provides adapters in several popular languages and several debugging tools and services. Also, to check whether everything is up-and-running smoothly, it also has an admin tool.

## 1.2 Scope of the Test-bed

This document limits its scope to the core Test-bed design, more specifically, the design of the Test-bed's reference implementation, which is an implementation of the [Test-bed specification](#) (see Figure 5): it thereby provides an overview of the most important components of the Test-bed, how they work together, and how they can be used by different stakeholders.

The [CM](#) solutions and simulators that supplement the Test-bed, however, are *not* part of the Test-bed. The simulators' function is to simulate an incident, and the reactions that may occur in a real world, since we cannot unleash incidents like a flooding and earthquakes on the real world. The solutions are the actual tools that are trialled and evaluated, and measured whether they actually do solve a [CM](#) gap. These solutions are fed with the simulator's output, and perhaps the output of other solutions, so end users can observe and evaluate their contribution during a fictive incident.

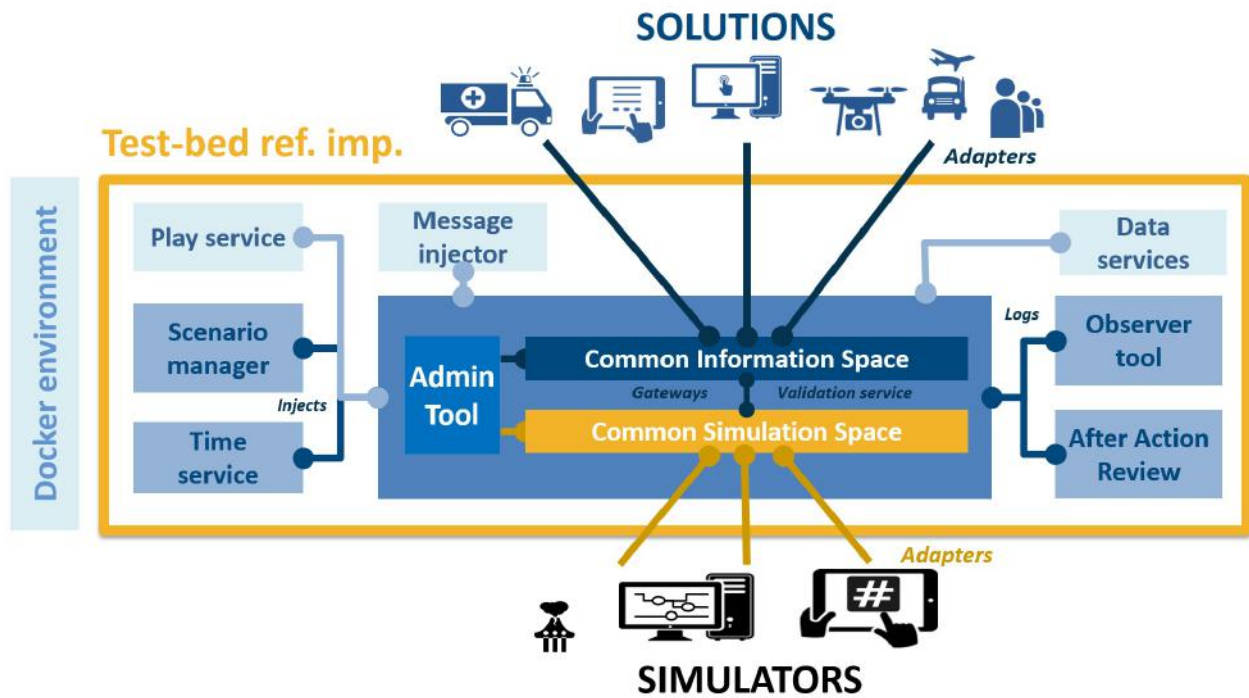


Figure 5. Scope of the test-bed.

## 1.3 Organisation of the Document

This is a live document, and the latest version can always be found online at [www.gitbook.com/book/driver-eu/test-bed-design](http://www.gitbook.com/book/driver-eu/test-bed-design). It is organised as follows:

The [Test-bed description](#) provides a general overview of the Test-bed reference implementation. It is an easy to read chapter which requires little technical knowledge, and is aimed at anyone who has to work with the Test-bed.

The chapter, [Test-bed for Trial owners](#), is specifically aimed at [Trial](#) owners and practitioners, and discusses the functionality the Test-bed offers to them. And also, what it does not offer.

In the chapter, [Test-bed for developers and sysops](#), the technical side of the Test-bed is explained. Specifically: how to manage a Test-bed as a sysop, or how to connect a solution or simulator to it as a developer.

The final chapter offers more details about the [Test-bed design](#), and provides an explanation for the main design decisions. The intended audience are developers that need a deeper understanding of the Test-bed and its underlying architecture, e.g. for supplementing the Test-bed or for offering a deeper integration of their solution.

## 1.4 What's new

## **Version 1 (2018-03-31)**

This is the first version of D923.21 Test-bed reference implementation v1. It matches the release version of the software applications as available on GitHub: [github.com/DRIVER-EU](https://github.com/DRIVER-EU). Two future releases of this document are planned, v2 and v3, and the major changes between the different documents will be described here. In the mean time, this online documentation is continuously updated, so it matches the current state of the Test-bed.



## 2. Test-bed description

The Test-bed supports practitioners by providing an environment in which they can easily [Trial](#) new solutions and run exercises. In this chapter, the main components of the Test-bed are explained (see Figure 6).

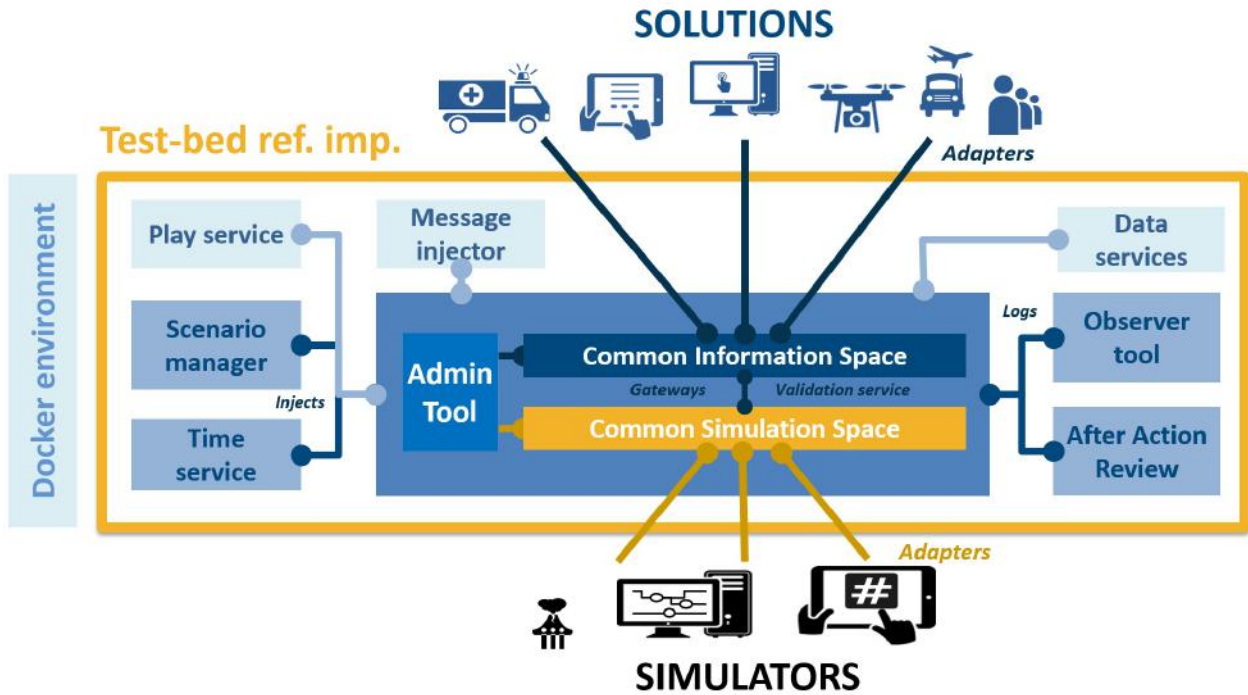


Figure 6. Explanation of the Test-bed components.

## 2.1 Core

The Test-bed must support the exchange of information between distributed solutions, simulators and tools. Information such as the location of an incident, alert messages, or the locations of vehicles. Comparable to people exchanging information via email, chat or twitter, the Test-bed exchanges information using the open-source messaging system [Apache Kafka](#) from the [Apache organisation](#).

As it is assumed that the systems that are connected via the Test-bed are either software systems, or hardware with a software interface, the Test-bed's support for non-technical systems is limited: typically, support will be limited to the evaluation tools, such as the Observer Support tool and After-Action Review tool.



# Adapters

Being widely used worldwide, Kafka has connectors for most programming languages, so software applications can easily connect to it. While connected to Kafka, and therefore the Test-bed, the application can send and receive messages. When you want to receive a message, you subscribe to a topic of interest: thereafter, you get all the messages that are sent instantaneously until you end your subscription. Optionally, you can even get messages that were sent in the past, or while you were offline, as Kafka logs all messages for a pre-set time. To publish a message, you just need to send it to your topic of interest.

*For example, to send a CAP (Common Alerting Protocol) message to all interested parties, you use a connector to send your CAP message to the 'cap' topic. Every tool that has subscribed to the 'cap' topic will get it right away.*

The default Kafka connectors are lacking certain features that are useful in a Test-bed environment, so some existing connectors have been extended. These extended connectors are called **adapters**, and the Test-bed currently maintains four of them: in [Java](#), [C#](#), [JavaScript/TypeScript](#) and [REST](#). Note that the [REST](#) adapter is a simple interface so any application can send and receive messages using basic internet commands.

Adapters extend regular Kafka connectors with:

- *Heartbeat signals*: Before you can start a [Trial](#), every solution, simulator and tool needs to be up-and-running. Therefore every adapter transmits a heartbeat signal every 5 seconds to inform others it is online.
- *Logging*: Besides being online, it is also important to know that each connected service is running as expected, so each adapter offers the option to log warnings/errors to the Test-bed as well.
- *Configuration options*: The adapter can inform others to what topics it subscribes and publishes. In addition, this can be configured too externally.
- *Time*: A [Trial](#) scenario typically will not run at real-time, so the adapter needs to share the fictive simulation time. In addition, it shares the simulation speed, as we may be running slower or faster than real-time, as well as the simulation state.

# Messages

As software applications need to understand the messages they receive, the Test-bed has to assure that every message that is sent complies with the expected format (syntax). For example, when a solution wants to share the location of a vehicle or the value of a sensor, you probably need to capture the vehicle's or sensor's location, as well as its type, speed or

sensor value. Then it is important to know that the type will be one out of a list of possibilities, that the location is specified using two numbers, and that the speed or sensor value is a number too.

To capture this information, the common solution is to specify it in a so-called schema. The Test-bed enforces this too, and it uses the open [Apache AVRO](#) schema format.

**Dealing with standards:** In the [CM](#) domain, several standards exist, such as CAP, EDXL or EMSI. They are represented using XML, a textual representation of a message that is easily readable by computers. A recurring problem with all standards, however, is that they rarely represent all the information you would like to share. This often leads to adding new fields, or, even worse, *re-purposing* existing fields. Additionally, not every organisation uses it in the same way. For Trialling new solutions, the Test-bed needs to be flexible and exact, and that's why the Test-bed does support these standards, but converted to the [AVRO](#) format. In that way, every connected solution or simulator will exactly know what to expect when reading a message, as new fields can be easily added in a robust way.

## CIS and CSS

At the heart of the Test-bed, i.e. its [core](#), all messages are exchanged using [Apache Kafka](#). Conceptually, though, we distinguish between a Common Information Space ([CIS](#)) and a Common Simulation Space ([CSS](#)). The [CIS](#) is where the solutions exchange information, and the [CSS](#) is for simulators. Typically, the [CIS](#) will exchange fewer messages during a [Trial](#), and time synchronisation is simple. In the [CSS](#), many more messages are generated, e.g. the location of all vehicles may be updated every second. Simulators may need to be in sync with others, e.g. a flooding simulator may flood an area, and at the same time, the traffic in the same area should experience the flood too.

For simple Trials, the [CSS](#) and [CIS](#) will run in the same Test-bed. In case the performance suffers, it may be necessary to split the [CSS](#) and [CIS](#) over two test-beds that are interconnected.

Note, though, that the [adapters](#) can be used to connect to the [CIS](#) as well as the [CSS](#), so there is no difference between them.

In rare cases, the [CSS](#) may be replaced, or extended, by one of the existing simulation standards such as [HLA](#) or [DIS](#), that are especially popular in the Defence sector. Please refer to Chapter 4 to learn more about this.

## Gateways and Validation Services

Even while using well defined messages based on [Apache AVRO](#), it is certain that not all solutions and simulators speak each other's 'language'. Like in Europe, as not everyone is speaking Esperanto or English and there is a need for translators, in the Test-bed we need *gateways* to translate one topic's message to another. Examples are not only translating one message format to another, but for example to translate:

- A message from a simulator sharing the location of all vehicles, to a [COP](#) tool message that only contains the location of its own resources
- An EDXL Resource Management request from a [COP](#) tool to a simulator message, which in turn sends out an ambulance to the required location.

In order to facilitate solution tools to obtain their needed information from the simulated world, the [CSS](#) needs to be connected with the [CIS](#) by means of translator applications, residing in the [CIS-CSS](#) Gateway. These translator applications form the bridge between the simulated truth and the perceived/communicated truth by mapping the relevant changes from the simulated world to messages globally understood in the [CIS](#).

An example of this would be a simulation of a flooding. Imagine a river that has a rising water level due to increase of rain water. At the river bank, there are several sensors that react to the amount of water coming in contact with the sensor. An application is created and connected to current operational systems to send CAP messages regarding the water level in clear categories ranging from LOW to DANGEROUSLY HIGH. A possible solution is assessed on improving decision-making based on the messages outputted by the created sensor application.

In this example, the water in the river is handled inside the [CSS](#) by means of a simulator focussed at calculating water levels at exact locations. The sensor application would be a translator application or gateway, mapping the current water levels obtained from the [CSS](#) into the messages with understandable categories (and, for instance, only sending them whenever a change in category is observed) similar to the operational application. The tool connected to the [CIS](#) listens to the formatted messages of the translator application as if it was connected to the actual operational application.

There are also tools that will send out messages that serve as commands or requests to change the simulated world (e.g. Command & Control systems used to trigger procedures to be executed by units, which in case of a [Trial](#) are simulated). Again, a gateway would be used to bridge the two spaces together. For example, a new dispatch centre solution (i.e. a type of [COP](#) system) that allows the user to send out emergency services from their dispatch towards the incident location. The solution would send out a standard resource management message via the [CIS](#). The gateway service picks up the message and maps it towards the corresponding request for changing the simulation space. The responsible simulator would receive this request via the [CSS](#) and handles it, changing the respective simulation entity (i.e. letting a simulated unit drive in the simulated world).

**Validation Services** are specific gateways that, as the name suggests, validate a message in more detail, before it is passed on to other systems. For example, if application A is publishing a CAP (Common Alerting Protocol) message for application B, i.e. A --> CAP topic --> B, the Test-bed will make sure that it complies with the appropriate schema before passing it on. However, there may still be certain aspects in the message that are not completely correct, e.g. the alerting area that is represented as a polygon may not have the same starting and ending point (i.e. it should be closed), or the incident location that is represented by two numbers (x, y), may actually be published as (y, x). So during testing, the validation service can 'intercept' messages between A and B and validate them in detail. Only valid messages are passed on, i.e. A --> CAP validation topic --> CAP topic --> B.

## 2.2 Test-bed administration tool

The Test-bed is a collection of distributed services running in a network environment. You can compare it to a theatre play, where the stage needs to be prepared, the musicians must be ready, as well as the light and sound engineers. The Test-bed admin tool (see Figure 7) helps you by monitoring both the [CIS](#) and the [CSS](#) to support understanding what is/was going on during a [Trial](#): to determine whether all services are ready, and that their inputs and outputs are correct. Also in case a service encounters any errors, this is made visible and the errors can be inspected - are they serious and do we need to stop our [Trial](#), or can we ignore them safely and run on.

This does not only apply to the Test-bed's core tools, gateways and services, but also for the connected simulators and solutions.

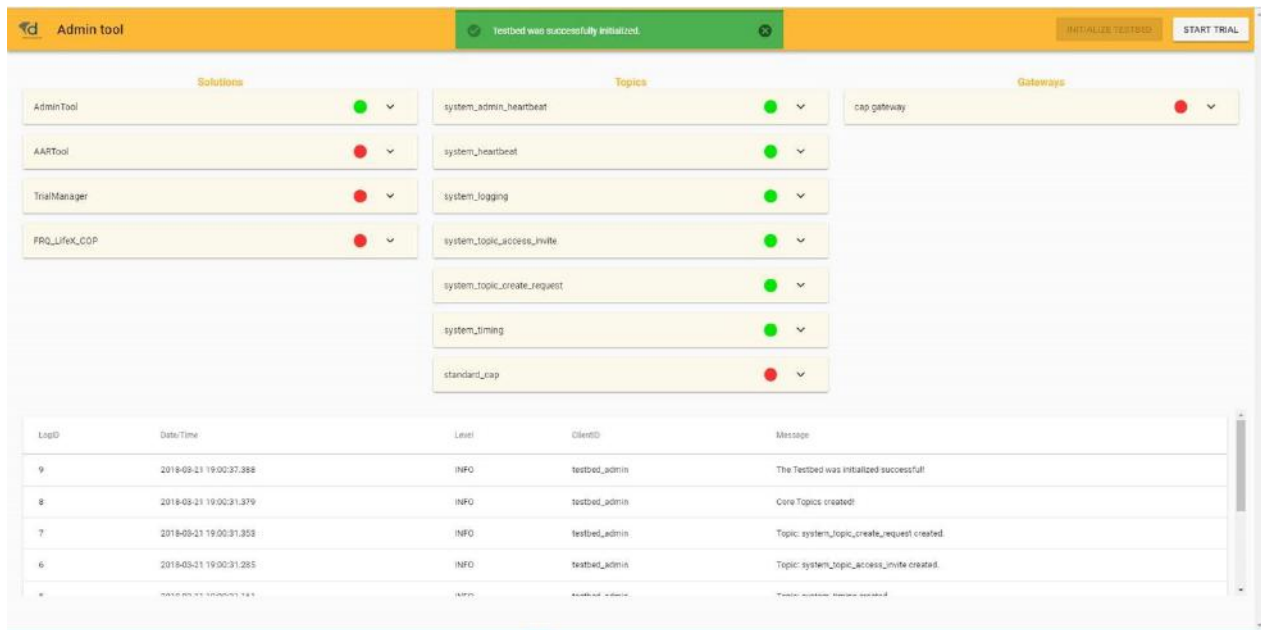


Figure 7. Admin tool.

Detailed information:

- [Functional specification](#)
- [Website](#)

## 2.3 Trialling, Exercising and Scenario Management

Whether designing a **Trial** to evaluate solutions, or an **exercise** to train people, a scenario and, optionally, simulations, are needed to emerge the training audience and to give them the feeling that they are dealing with an actual crisis.

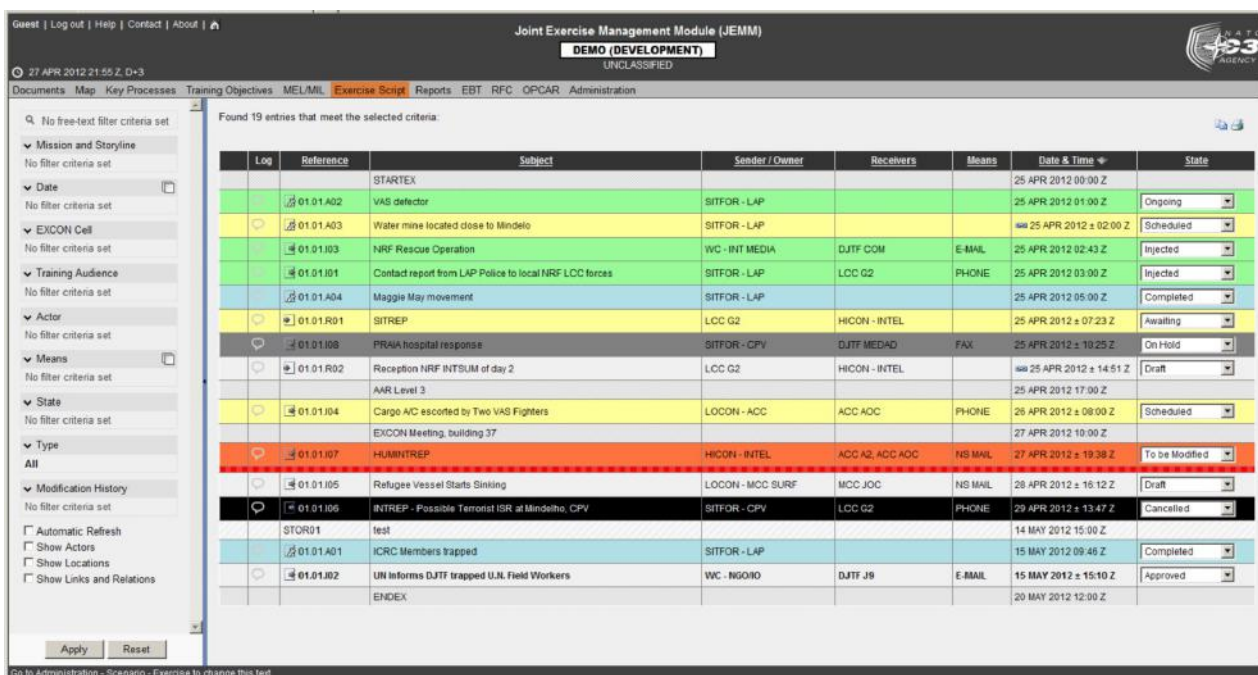
For an **exercise**, one starts by defining the *training objectives*, *What does the training audience need to learn?*. Next, an appropriate *Mission / Main scenario* is formulated in which these training objectives can be tested and exercised. The mission is further broken down into *storylines*. A Storyline describes a developing situation that will set conditions and provide the Training Audience an opportunity to achieve a specific Training Objective and optional secondary Training Objectives. It often targets a subset of the training audience, e.g. only the fire fighters, and consists of timed events, or so called *injects*. Think of an email to the commander, a 'start flooding' message to a flooding simulator, or instructions to a role-playing actor.

In a **Trial**, although the primary objective is not to train people, but to test and evaluate solutions, still a similar procedure can be followed. In that case, the *training objectives* are replaced by *research questions*, but the other steps remain the same.

## Existing software

The process described above is the typical approach taken by the NATO Joint **Exercise Management Module (JEMM)** (see Figure 8). It is a tool to support live exercises as well as table top exercises, from a few people to battalions. It puts a lot of emphasis on authorization management (*who can do what?*) during the creation of a scenario, and has a limited level of automation. For example, JEMM can connect to Outlook / Exchange Server to automatically create or receive email messages. Sending, though, is still a manual process.

JEMM is available free of charge to military NATO members, i.e. to use it in a **Trial**, a military presence is required.



Log	Reference	Subject	Sender / Owner	Receivers	Means	Date & Time	State
		STARTEX				25 APR 2012 09:00 Z	
	01.01.A02	VAS detector	SITFOR - LAP			25 APR 2012 01:00 Z	Ongoing
	01.01.A03	Water mine located close to Mindelo	SITFOR - LAP			25 APR 2012 ± 02:00 Z	Scheduled
	01.01.I03	NRF Rescue Operation	WC - INT MEDIA	DJTF COM	E-MAIL	25 APR 2012 02:43 Z	Injected
	01.01.I01	Contact report from LAP Police to local NRF LCC forces	SITFOR - LAP	LCC G2	PHONE	25 APR 2012 03:00 Z	Injected
	01.01.A04	Maggie May movement	SITFOR - LAP			25 APR 2012 05:00 Z	Completed
	01.01.R01	SITREP	LCC G2	HICON - INTEL		25 APR 2012 ± 07:23 Z	Awaiting
	01.01.I08	PRAIA hospital response	SITFOR - CPV	DJTF MEDAD	FAX	25 APR 2012 ± 10:25 Z	On Hold
	01.01.R02	Reception NRF INTSUM of day 2	LCC G2	HICON - INTEL		25 APR 2012 ± 14:51 Z	Draft
		AAR Level 3				25 APR 2012 17:00 Z	
	01.01.I04	Cargo A/C escorted by Two VAS Fighters	LOCON - ACC	ACC AOC	PHONE	26 APR 2012 ± 08:00 Z	Scheduled
		EXCON Meeting, building 37				27 APR 2012 10:00 Z	
	01.01.I07	HUMINTREP	HICON - INTEL	ACC A2, ACC AOC	NS MAIL	27 APR 2012 ± 19:38 Z	To be Modified
	01.01.I05	Refugee Vessel Starts Sinking	LOCON - MCC SURF	MCC JOC	NS MAIL	28 APR 2012 ± 16:12 Z	Draft
	01.01.I06	INTREP - Possible Terrorist ISR at Mindelo, CPV	SITFOR - CPV	LCC G2	PHONE	29 APR 2012 ± 13:47 Z	Cancelled
	STOR01	test				14 MAY 2012 15:00 Z	
	01.01.A01	ICRC Members trapped	SITFOR - LAP			15 MAY 2012 09:46 Z	Completed
	01.01.I02	UN informs DJTF trapped U.N. Field Workers	WC - NGORO	DJTF J9	E-MAIL	15 MAY 2012 ± 15:10 Z	Approved
		ENDEX				20 MAY 2012 12:00 Z	

Figure 8. JEMM exercise script example.

Alternative commercial solutions exist too, such as **Exonaut** (see Figure 9). They also aim at a military audience, and follow a similar approach.



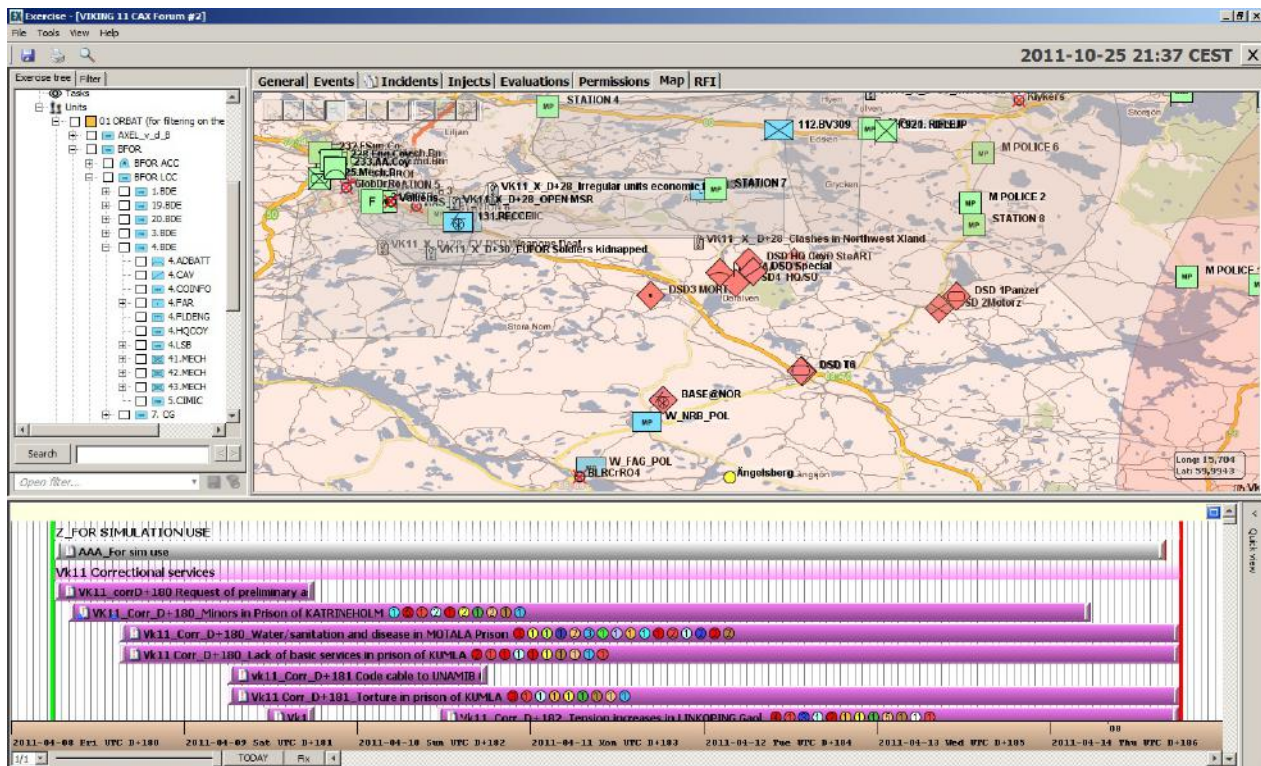


Figure 9. Exonaut timeline example.

## Scenario manager

A [Scenario Manager](#) is an integral part of the Test-bed reference implementation too, since it is not possible to use JEMM or Exonaut directly, as:

- JEMM is only available to NATO members, and can only be used in a an [exercise](#) when military personnel requests it. This will not always be the case.
- JEMM and Exonaut are aimed at the military community, and the fit with the Crisis Management domain is not optimal.
- JEMM and Exonaut are closed source, so a strong integration with the Test-bed is not possible, as the applications cannot be modified.

The Test-bed's [Scenario Manager](#), then, acts as the *composer* and *conductor* of a classical performance:

- As the *composer*, it defines what each role has to play. For example, what do the simulators or role-players need to do in order to provide a realistic incident and background to the [Trial](#), or it could include sending direct messages to solutions.
- As the *conductor*, it controls *when* each role starts and stops.

Additionally, the [Scenario Manager](#) will also publish messages that are not directly related to the scenario itself. For example, it can send a message to the observers, informing them that they need to pay attention, as something important is going to happen soon. Or it could

ask these observers specific questions during the [Trial](#), e.g. '*Did you notice that X occurred?*'. These messages are also important for the after-action review, as they can be used as bookmarks to quickly go to parts in the scenario that are of extra importance for the evaluation.

Detailed information:

- [Functional specification](#)

## 2.4 Evaluation

Evaluation is needed to verify that the [Trial](#) objectives have been achieved. The Test-bed provides two services for this: an Online Observer Support tool and an After-Action Review tool.

### Online Observer Support tool

Based on the specified objectives of the [Trial](#), an observer expects to observe different kinds of behaviour. At the same time, there is little time during a [Trial](#) to record behaviour, as the [Trial](#) runs on, and that's why the [observer tool](#) provides [Trial](#)-specific pre-made forms (templates) to quickly create a new observation. For example, *Did you observe role X do Y?* Yes/No. These [trial](#)-specific forms are created before the [Trial](#) by the observation team manager in the administration panel. based on the data collection plan (as described in the [Trial](#) Guidance Methodology). Using this panel, specific forms can be assigned to specific observers. The observer can use a tablet, phone or desktop application for his work.

Although the [observer tool](#) (see Figure 10) can run standalone, outside of the Test-bed context, there are several benefits when it is connected, since this allows:

- To share observations with [Trial](#) staff: they can use this information to steer the [Trial](#) in a particular direction.
- The After-Action Review tool can use the observations during the analysis and evaluation
- The [Scenario Manager](#) can inform the observers of major events that are about to occur: so they are warned ahead of time
- Observation forms can be created dynamically and transmitted to selected observers

Although the [observer tool](#) enables the collection of personal data, research ethics is outside the scope of this technically-oriented document, and is being described in more detail in D922.21 - [Trial](#) guidance methodology and guidance tool specifications (version 1).



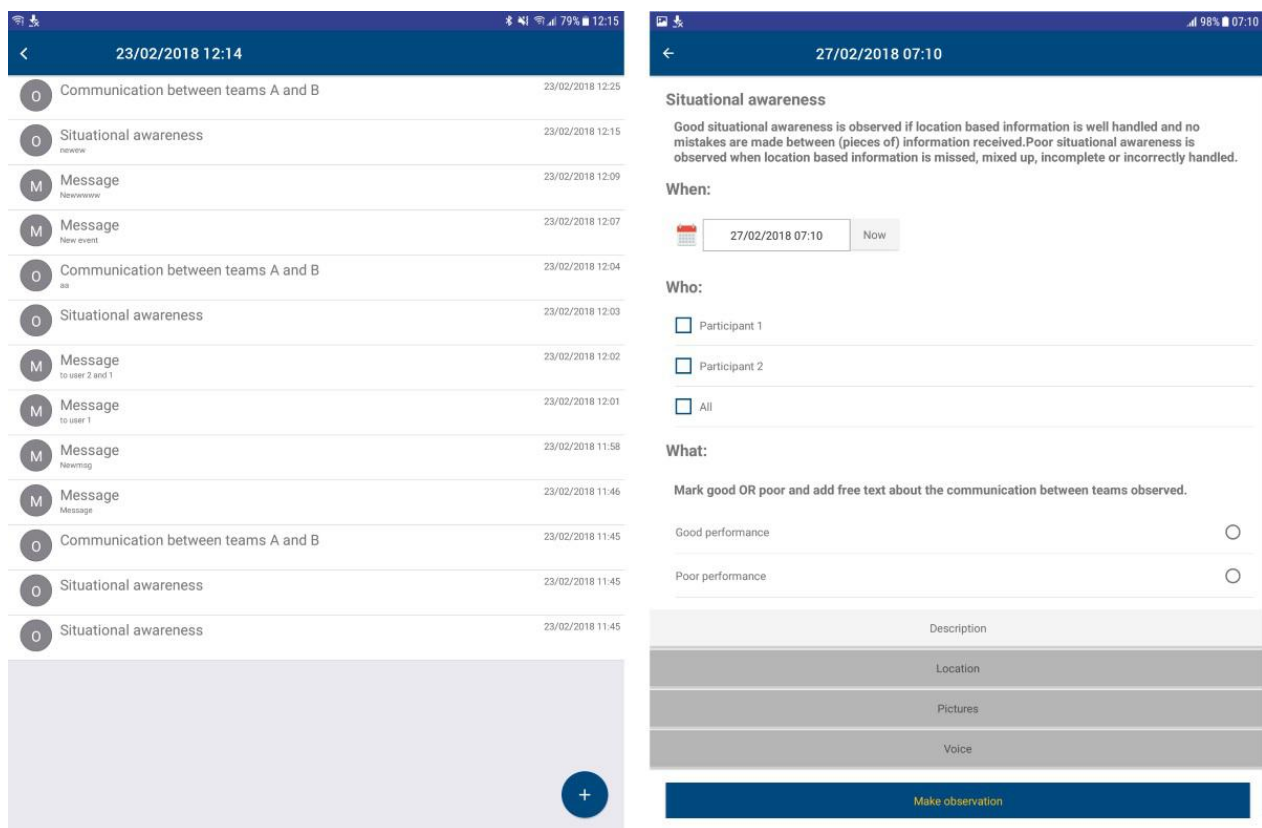


Figure 10. Observer Support Tool: Left, an overview of available observation templates. Right, one of the observation templates is selected.

Detailed information:

- [Functional specification](#)
- [Website](#)
- [Documentation](#)

## After-Action Review tool

The After-Action Review ([AAR](#)) tool provides the possibility to collect data after a [Trial](#) has finished and analyse it. Its main purpose is to facilitate the evaluation of the trialled solutions against the predefined objectives, and to help the participants determine how well they functioned. It collects messages (exchanged during [Trial](#)), observation reports and takes screenshots of the applications during their use.

Detailed information:

- [Functional specification](#)

## 2.5 Simulation

Much can be said on the subject of simulation, but for the purpose of this chapter, it suffices to provide a brief overview of the Test-bed's relation to simulation.

In the Test-bed, the goal of simulation is to provide a realistic, immersive background for the [Trial](#). Typically, this requires:

- A **simulation of the incident** e.g. a flooding, earthquake or explosion, etc. simulation
- A **simulation of the reactions** to the incident, e.g. people running away or drowning, buildings collapsing, road jams or traffic accidents, etc.
- A **simulation of the perceived world**, i.e. painting a picture of the world to solutions of what they are reasonably expected to see, not what is actually happening. For example, when an area is flooded in the simulation, all simulators know the exact location of the water. So if people are standing knee-deep in the water, or a road is inaccessible due to water, that can be shown and used. However, a [COP](#) tool or other solution does not have such a perfect view of what is happening in the world. It does not know where everyone is, nor the exact location of the water level. As long as it has no sensors, cameras, drones, or people informing it, it may well believe that the flooding is in an entirely different location or not happening at all. In a recent [CM exercise](#), it took the participants quite some time to figure out that the water was actually threatening their own location, and they hadn't taken the necessary precautions. A well-designed [Trial](#), therefore, needs to think about how they are going to present the simulated world within the [Trial](#).

The Test-bed, therefore, offers support to simulators for creating this realistic and immersive background, by:

- Providing a time-service: i.e. each adapter knows the scenario time, so simulators and solutions can use this in their user interface and calculations. Think of a clock display, but also when sending an email or CAP message, making sure it uses the correct timestamps.
- The [Scenario Manager](#), as discussed above.

It does not, however, provide these simulators as an integral part of the Test-bed. They are, and shall always remain, external. Even though some simulators will be connected during the project, they are external, as as such, also not bound by the open source requirements that the Test-bed has to adhere too. For example:

- XVR connects their 3D crisis management environment, Crisis Media and Resource Manager to the Test-bed, thereby offering their (commercial) services to other parties too.
- DLR connects their open source SUMO (Simulation of Urban Mobility) traffic simulator to the Test-bed, which provides realistic traffic during an incident
- Thales connects their commercial Crowd Simulator to the Test-bed, e.g. providing a

realistic simulation of people in need during a crisis.

### **A note about Simulators**

All simulators have their own data model of how they represent the simulated world. The [CSS](#) allows these simulators to agree on a communication form that the simulators understand to create and maintain a jointly simulated world.

The simulators only need to be concerned with maintaining the current state of a given location (including entities and processes present at that location), and do not have to deal with the different kinds of communication types for tools and users to depict that current state.

The [CSS](#) allows simulators to only focus on maintaining the current state of the simulated world (i.e. the simulated truth of the incident and the world around it). In order to communicate state changes with other simulators inside the [CSS](#), self-created communication messages are allowed inside this space. This is different than the messages being sent over the [CIS](#), because the [CIS](#) is more aligned with current emergency management standards (like Common Alerting Protocol (CAP) messages, or Emergency Data Exchange Language (EDXL) messages).

## 3. A Test-bed for Trial Owners

**Trial** owners will interact with the Test-bed when they want to **Trial** one or more solutions. Since Trials vary in complexity and scope, not all **TGM** steps may be required for every **Trial**. On the one side of the spectrum, the **Trial** may be more like a simple and inexpensive software **test**, in which a **Trial** Owner can verify that the solution has useful features, is user-friendly, looks good, and doesn't crash. Typically, this would be done with only a few people. For a **CM** solution, such a **Trial** also requires (some components of) the Test-bed, together with a test-scenario. The test-scenario allows you to test the solution in a relevant context.

For example, if one would test a **COP** tool as is, it would probably not have any map layers, basically being an empty map. Using a test-scenario, the **COP** tool can be tested in the context of a simple incident, perhaps a flooding, have some ambulances driving around, and map layers showing population information, locations of critical infrastructure, etc.

On the other side of the spectrum, a **Trial** could involve multiple solutions, many participants and observers, in a more realistic, operational-like scenario, with proper data collection and evaluation along all dimensions in place. In this case, the full **TGM** should be applied to perform a proper **assessment** and funded conclusions can be drawn. As the latter requires a serious investment in time, people and budget, a **Trial** Owner may consider to combine the **Trial** with a planned **exercise** to reduce the costs of preparing, executing and evaluating a **Trial**.

In the following sections, a use case (storyline) description of each is provided. The main actor in each story is Monica, a regional crisis manager with a professional background in fire-fighting. One of the challenges she is facing is that she does not always have a good overview of where her people and trucks are during a large-scale fire, and she is looking for a solution.

### 3.1 Use Case: Evaluating a solution standalone

Monica has heard about an interesting **COP** solution, *csCOP* via the DRIVER+ Portfolio of Solutions. She considers using it to address her problems

1. Monica visits the Test-bed's **composer website** (see Figure 11).
2. She briefly reads its homepage, which explains her what to do next.
3. She opens the solution's tab and sees that *csCOP* is available for evaluation.
4. She select *csCOP* in the menu (see Figure 12).

5. She understands that it is difficult to test a solution without any data / scenario, so she visits the data tab and selects a fire-fighting data set situated in the South of France: *It involves a large-scale forest fire, which is rapidly spreading. Ambulances and fire trucks are deployed and driving around. She can also look at census data of the area and a weather report.*

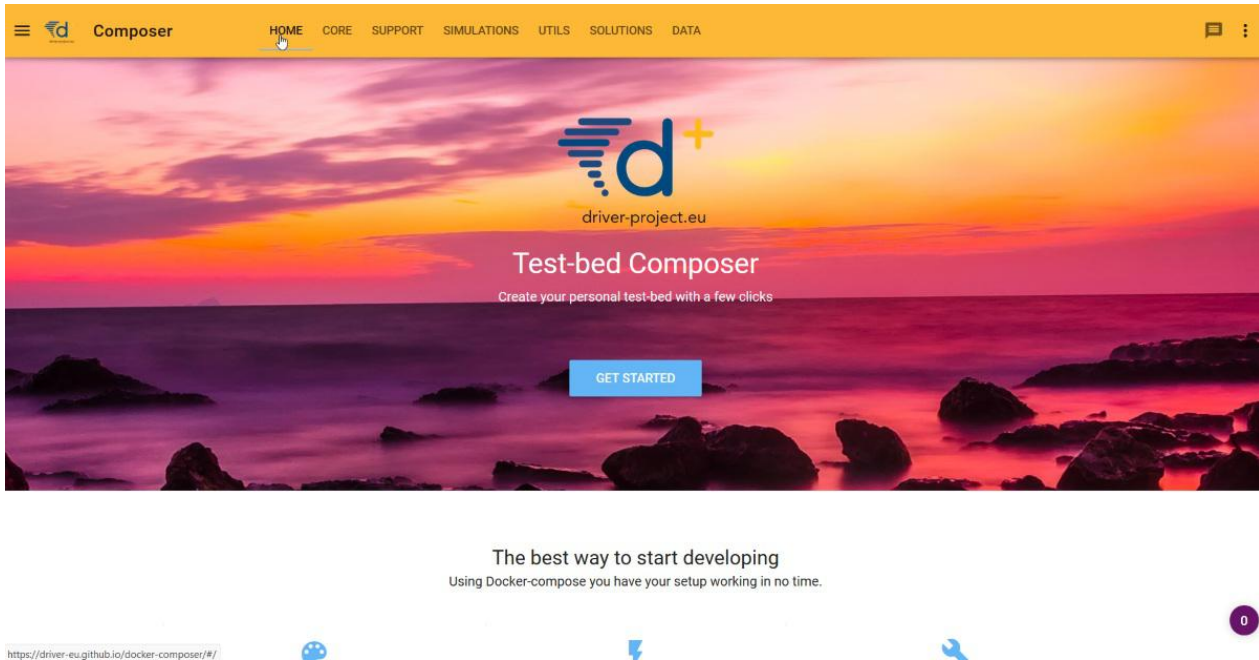


Figure 11. Test-bed composer's home page.

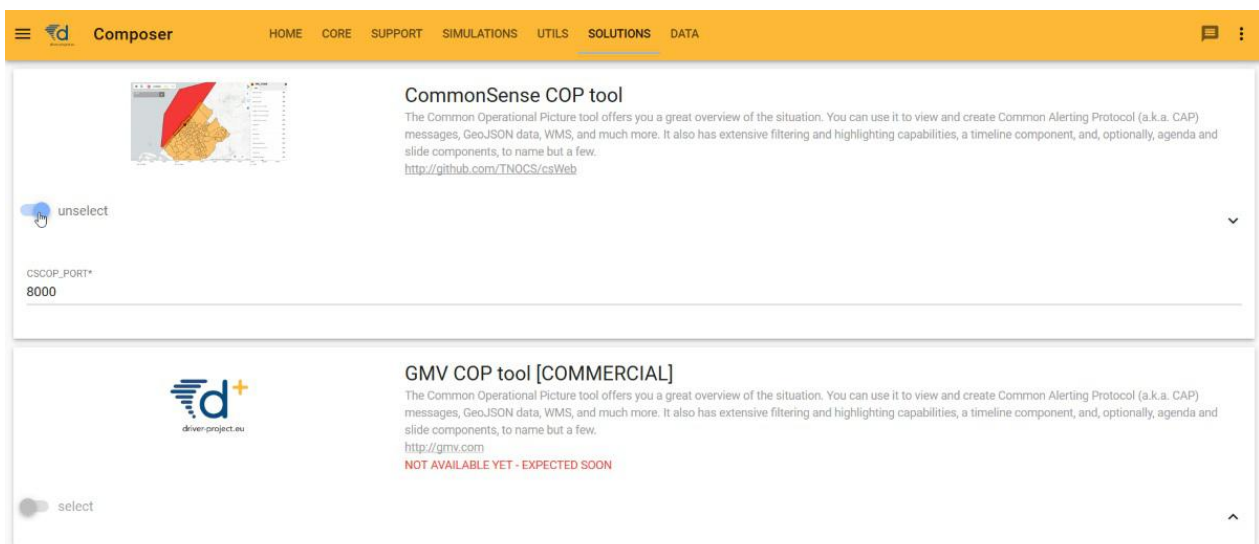


Figure 12. Test-bed composer: Selecting a solution.

*From here on forwards, two alternatives are presented. The first one represents the desired situation. The second alternative is already in place.*

## Alternative 1: Cloud scenario

1. As she currently isn't interested in other solutions, she opens the menu and clicks on the DEPLOY button. A dialog opens and informs her to wait while her selected Test-bed is started in the cloud.
2. After a minute or so, the Test-bed is running and she gets her own unique link. She visits this website, and is presented with a simple menu: she can start (pause | stop) the fire fighting scenario, and open the website of the *csCOP* tool. *In its layer menu, she can turn on the layer which shows the ambulances and fire trucks. There are also options to turn on the fire fighting layer to show the location of vehicles and staff, etc.*

## Alternative 2: Local scenario

1. As she currently isn't interested in other solutions, she opens the menu and clicks on the BUILD button. A dialog opens and she can download the *Docker-compose* file to her PC (see Figure 13).
2. Running a simple command, the Test-bed is downloaded and started on her own PC, and she can interact with the Test-bed as described above.

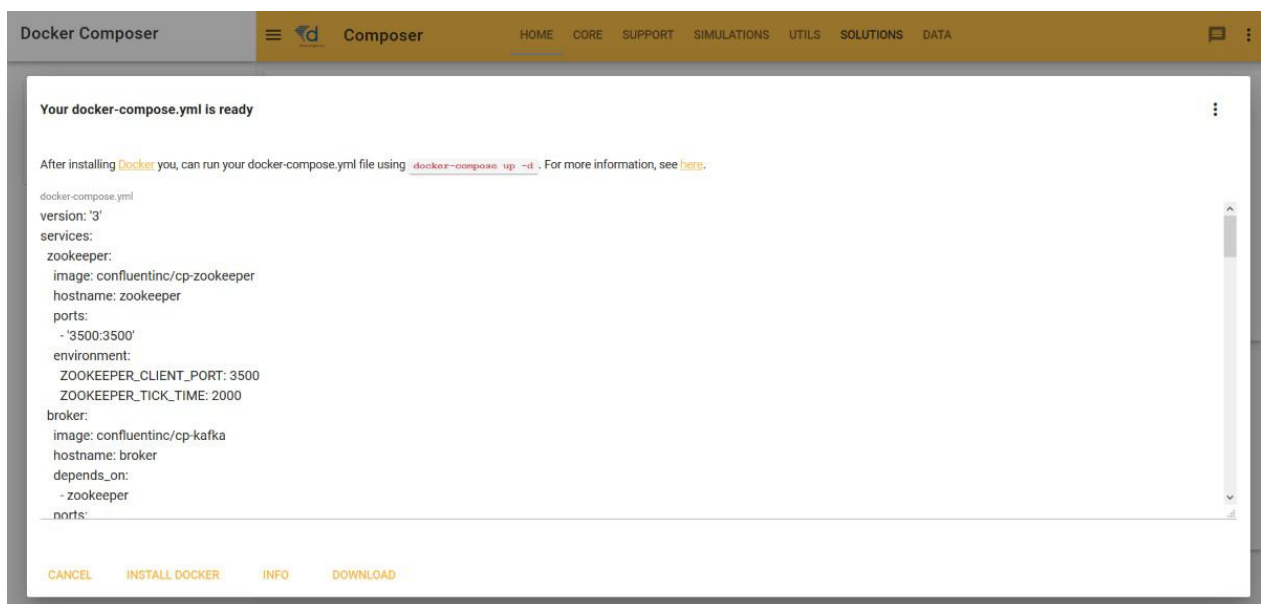


Figure 13. Test-bed composer: Downloading the *docker-compose.yml* file.

## 3.2 Use Case: Assessing solutions using a Trial

In order to run a [Trial](#), the same process as described above is followed. The main difference is that more services and solutions will be added, and in many cases, perhaps for security reasons, the Test-bed is run locally on the internal network. Basically, instead of only using the Test-bed core's services, a more complete Test-bed is required, also containing the [observer tool](#), [AAR](#), and [Scenario Manager](#).

In addition, the Test-bed is needed already well-before the final [Trial](#) date, since:

- The scenario must be created iteratively.
- Not all solutions or simulators are already able to connect to the Test-bed. So even before the first dry-runs, solutions and simulators should be able to test their connection and integration.

Typically, setting up the Test-bed for a [Trial](#) would not be performed by the [Trial](#) owner herself. Most likely, she will be supported by a local system administrator or consultant to help her decide what she needs, and to prepare the technical infrastructure.

When the Test-bed is running, though, the [Trial](#) owner has to take responsibility again for defining the scenario in case she has not delegated this task. The scenario is key in that it translates specific objectives to a storyline so the solutions or people can be put to the test.

As the [Scenario Manager](#) is not available in the current version of the Test-bed, only a brief outline can be provided. During the preparation, the [Scenario Manager](#) acts as the composer:

- The objectives of the [Trial](#) are defined.
- An overall scenario is described that can put these objectives to the test.
- Specific storylines are created to stress the solutions.

During a [Trial](#), the scenario is started by the [Trial](#) owner. The [Scenario Manager](#), acting as the conductor now, instructs everyone when to perform their planned act. Optionally, she can control the scenario time, for example freezing the time during a break, or to steer the [Trial](#) in another direction.

## 4. The Test-bed for developers and sysops

A [sysops](#) (system operators or system administrators), in the current context, is responsible for installing the Test-bed on their local network, and making sure that all the solution and simulation providers can get access to this network as well. This task is discussed in the use case 'Installing the Test-bed'.

A developer would be tasked with connecting an existing solution or simulator to the Test-bed. Besides the direct coupling, allowing their tools to receive and publish messages, it most likely also involves translating existing messages to their own format. Finally, in case you are connecting a simulator, you also need more detailed information about the time management in the Test-bed.

### 4.1 Use case: Installing the Test-bed

The previous chapter already explained how to setup the Test-bed. More detailed information about how to run the [Docker](#)-compose environment can be found [here](#). Alternatively, in the near future, use the [GUI](#).

System administrators are also responsible for setting up the local network, such that all solution and simulator providers have access to the local intranet as well as extranet.

In particular, it should be considered that some providers make heavy use of the network, e.g. to download maps, stream video, or access external computer clusters. If that is the case, consider using a throttling service in your network, so one provider does not claim all the network traffic.

More directly related to the Test-bed, however, is the connection of all solutions and simulators: are they connected correctly to the Test-bed, do they run without errors, are they subscribed to the correct topics, and do they publish to the expected topics, are some of the questions that the admin tool can answer for you. In addition, the admin tool makes sure that all message schemas are available. And when everything is in place, the actual [Trial](#) can start. Finally, the admin tool offers a convenient interface to all the other technical Test-bed services, such as the [REST](#) services, Topics UI, Schema Registry, Kafka Connect, etc.

From then on, the system administrator only needs to check whether the Test-bed does not experience any issues, like disconnected applications.



## 4.2 Use case: Integration process for a single solution or simulator

Within DRIVER+, a dedicated integration process *for solutions* is described in a separate document, D934.21, "Solution testing procedure". This section describes how to integrate with the Test-bed reference implementation, and also covers simulators and other tools.

This use-case is executed during the integration of a single solution or simulator into the Test-bed. It does not apply to the testing of multiple applications, and should already have been successfully performed before the application is tested in a larger context with multiple solutions and simulators.

1. Developer starts up the Test-bed and the Admin Tool, or uses a Test-bed available online.
2. Developer selects one of the existing adapters, integrates it into his application, and connects to the Test-bed. The Admin Tool shows whether the connection is established successfully. Adapters are available in multiple languages: [Java](#), [C#](#), [JavaScript/TypeScript](#) and [REST](#). A Python version will be available in the next release.
3. Developer defines the input/output [AVRO](#) message schemas and topics based on the running test-bed. Many popular schemas for CAP, EMSI, MLP, GeoJSON etc. have already been defined in our [AVRO-schemas repository](#) and should be re-used if applicable.
  - i. In case your message format is not available, you need to create a new one and register it with the Test-bed's schema registry. You can do that manually, or alternatively, the adapter will do this for you. The registration procedure is a bit different for each adapter.
  - ii. In case your information is available, but in a format that the application does not support, you can create a gateway service to translate messages in one topic, e.g. MLP, to another message format, e.g. GeoJSON, and consume the latter.
4. Developer starts up the [Kafka replay-service](#) to send them one-by-one or replay a logged sequence of messages. To log the messages in a topic, you can use the [Kafka-topics-logger](#) or the topics UI to save them. This is, for example, useful when you need to integrate with an application that does not run locally, e.g. when your [COP](#) tool needs to consume messages from a simulator that you do not have running locally.

In the next version of the Test-bed reference implementation, there will also be a message injector application, comparable to [Swagger](#) or [Postman](#), in which you will be able to create your own messages using a friendly user interface. It will use the [AVRO](#) schema to automatically create a form for defining your messages.

5. In case the integrated application also sends messages, Developer can use the Kafka topics UI to verify that they were created and published to the Test-bed successfully.
6. When your message uses time, you need to query the adapter to get the local [Trial](#) time. A first version of the Test-bed's [time-service](#) to manage the [Trial](#) time has just been released, and some adapters already offer an interface to it. So there is no need to query the Test-bed yourself to get these messages. In case no time messages are available, i.e. you are not running a [Trial](#), it returns the local system time. Please also check the time management's state machine in [Section 4.5](#) below.

Currently, not many solutions have already been integrated with the Test-bed. Now that the Test-bed itself is stable, and adapters are available, the integration of many existing solutions will take place. In the mean time, not all solutions will be able to connect to the Test-bed as described above, and during an actual [Trial](#), the [Trial](#) coordinator has to decide if, and to what degree, a solution needs to be integrated.

## Message Topics UI

The Test-bed includes Landoop's [Kafka topics UI](#) service (see Figure 14) to inspect all the message topics that exist (default location <http://localhost:3600>). It can be used to inspect whether you have been successful in sending your messages to the Test-bed.

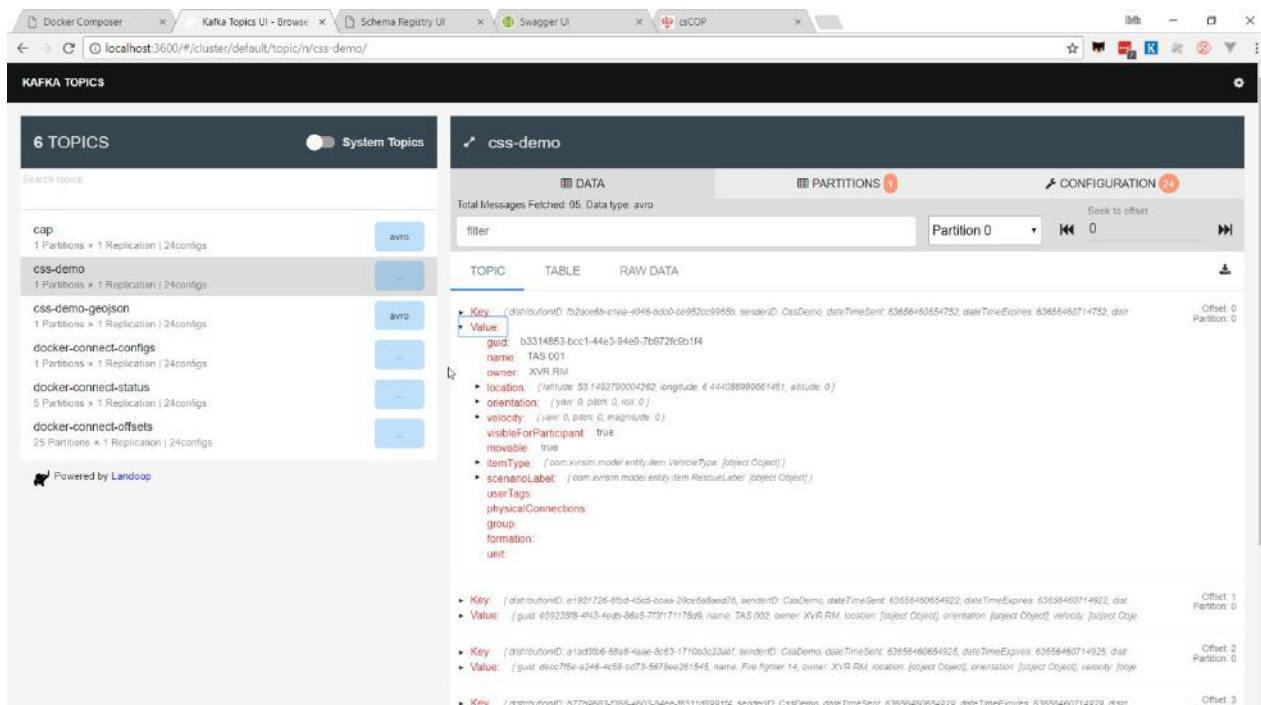


Figure 14. Screenshot of Landoop's Kafka topics UI, which is part of our test-bed.

## Schema Registry

The Test-bed also includes Landoop's [AVRO schema registry](#) service (see Figure 15) to inspect all the available schema's (default location <http://localhost:3601>). As each message topic only has one schema, every message send to a topic needs to comply with that schema too. Also, in case a developer is creating new messages, these schemas must first be added to the schema registry, either manually or via an adapter.

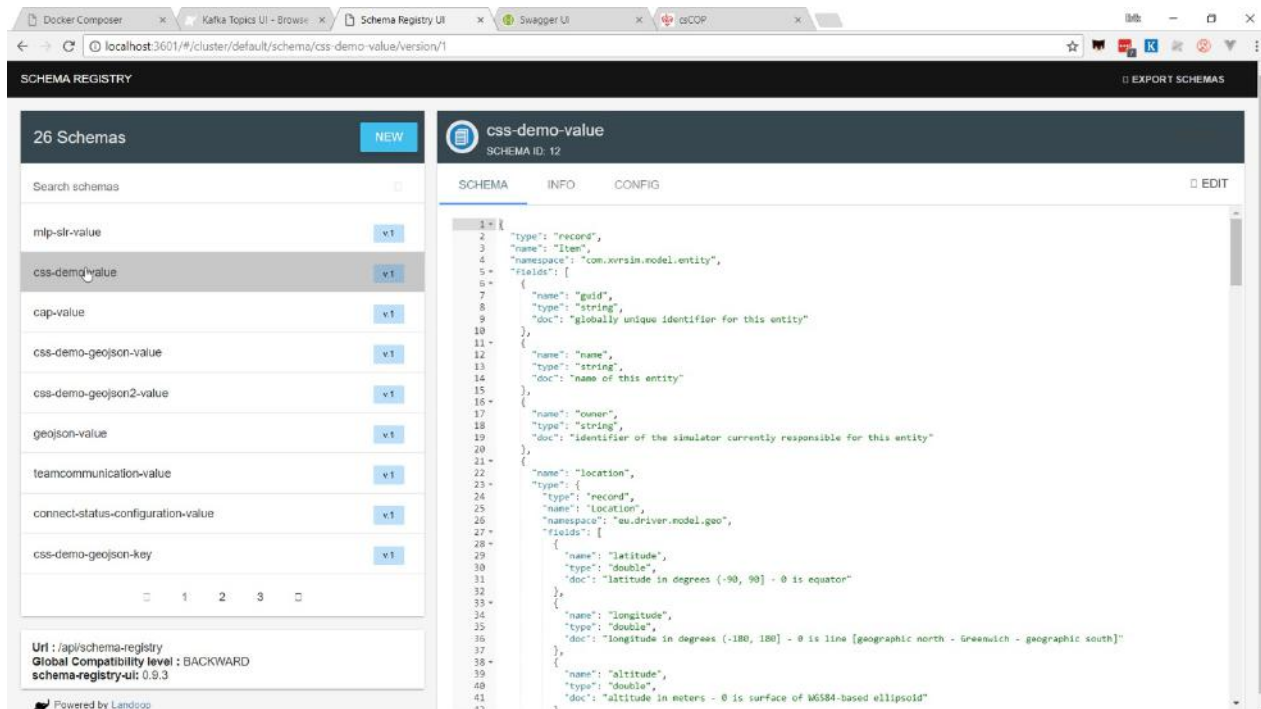


Figure 15. Screenshot of Landoop's [AVRO schema registry](#), which is part of our test-bed.

## REST service

The Test-bed contains a [REST](#) service: in case a (legacy) solution is not adaptable, or the solution is developed in a programming language that is not supported, it may be necessary to interact with the Test-bed via the [REST service](#).

## Replay service

Connecting to the Test-bed is needed to share information that you produce, or consume information from others. While working on integrating your own simulator or solution, however, it is very likely that there are no other simulators or solutions running. When sending messages, you can use the Kafka topics UI to verify that your messages have been delivered. And it is the purpose of the [replay service](#) (see Figure 16) to present you with messages to consume.

For example, assume that you as a developer are tasked to integrate a **COP** solution. It needs to consume the locations of the rescue vehicles, which are normally generated by a simulator. However, it is a commercial simulator that you do not have. In that case, you request the simulator to run a scenario and publish it to the Test-bed. Next, the simulator will log all the messages to file using the **Kafka-topics-logger**. This log file is subsequently sent to the **COP** solution developer, who can replay it using the replay service, as if it was the simulator was present.

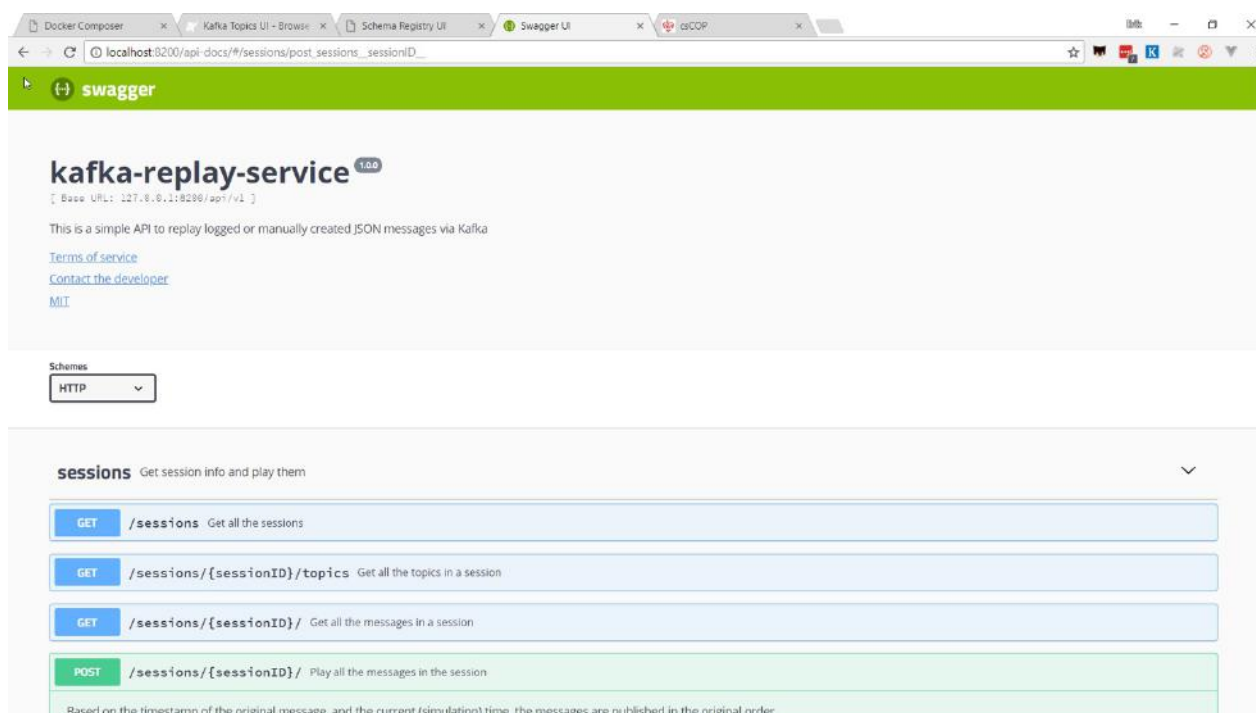


Figure 16. Screenshot displaying the Kafka-replay service's Swagger interface.

## 4.3 Use case: Pre-trial integration testing

The procedure for testing multiple solutions and simulators before an actual **Trial** with participants is performed, is similar to the procedure for testing a single application. It is assumed that the single solutions and simulators have already been successfully integrated with the Test-bed, and all required message schemas are defined.

1. Sysop starts up the Test-bed and the Admin Tool, or uses a Test-bed available online. If not done already, all required schema's are registered with the schema registry.
2. Sysop inspects the Admin Tool and verifies that all required solutions and simulators are available and running online without errors.
3. Sysop starts up the **Scenario Manager** (not yet available), loads the **Trial** scenario, and initializes it. The Test-bed's time service updates the fictive **Trial** time and state, and every application that uses time should reflect that.

4. Sysop runs the [Trial](#) scenario, either from the beginning or at another point in time, e.g. where issues were discovered. The time-service will update the fictive time accordingly.
5. Sysop resets the [Trial](#) scenario, and replays it, as many times as required to make sure that everything works as expected.

## 4.4 Gateways for translating messages

As a developer, you may be confronted with message formats you need to consume, but do not support natively in your application. In that case, you can either:

- Adapt your application to support these message formats natively.
- Create a gateway service which translates messages from one message format to a format that your application does understand.

To create such a gateway service is simple: you consume messages from one message topic, convert them, and publish them on another topic. The validation services follow the same approach, and several dedicated services are already available within the [DRIVER+ space on GitHub](#).

## 4.5 Data services and data sets

Within a [Trial](#), we need to create a virtual environment where we can safely experiment. This virtual environment is created using data, such as maps, census data, height data, power lines, cell towers, hospitals and care providers, etc. As it is a lot of work to create such a rich data environment, the effort should be shared among the [Trial](#) owners, solution and simulator providers. Not only to reduce the workload for a specific organisation, but also to make sure that all parties use the same data. Otherwise, a traffic simulator may use a different roadmap than the simulator that provides a 3D environment, and some roads may be blocked by buildings.

In many cases, real-world data is used, optionally enriched with scenario-specific information. Sometimes, a virtual environment is created, based on real-world data but with altered names.

So in order to share all this gathered data, the Test-bed offers two types of services:

- [Docker volume images](#) to store all this information together, so the data can be easily shared. A Test-bed user can simply pull the volume image from the [Docker](#) hub to have all data instantly available
- *Data services*, to share this data with all users, e.g. there is an [MBtiles service](#) to offer map images to [COP](#) and [COP](#)-like tools, or a [WMS service](#) that translate Test-bed

messages to WMS map layers available to make the information available to legacy systems.

**Security** is yet another reason to have these data services and data sets as part of the Test-bed. Not all Trials have open access to the Internet, but they still need access to this kind of data.

## 4.6 Time management

A **Trial** typically is not performed in real-time: either because the incident occurs at night, and people prefer to **Trial** during working hours, because of the limited availability of participants, or because it would simply take too long. An example of the latter is a flooding incident, which can start days before any flooding actually occurs, so you need to compress the scenario to normal working hours.

Within the Test-bed, therefore, the scenario time (a.k.a. **Trial** time or fictive time) is controlled via the **time service** (see Figure 17) using **two types of messages**: one for controlling the time, and one for informing adapters about the current scenario time.

As a developer, you do not need to interact with these messages directly, since:

- Every adapter has a time interface to get the current scenario time. Even as a solution developer, you should also use this time to timestamp the messages that you send. For example, if inside your message you refer to a particular time, always base it on the scenario time.
- Every adapter has a state describing the current scenario phase, which you can optionally use during the integration:
  - Idle: no scenario has started. The time interface returns the system time.
  - Initialized: the scenario is ready to be started. All adapters will receive the scenario start time, and can use this to initialize their service. In the near future, adapters have the ability to inform the Test-bed when they are initialized and ready to start.
  - Running (started or paused): the scenario time is moving forward, either in real speed or slower/faster than normally. In case the scenario is paused, the current scenario time is still actively being distributed, but does not progress (speed is 0).
  - Stopped: The scenario stops, and the simulation time is no longer being updated.

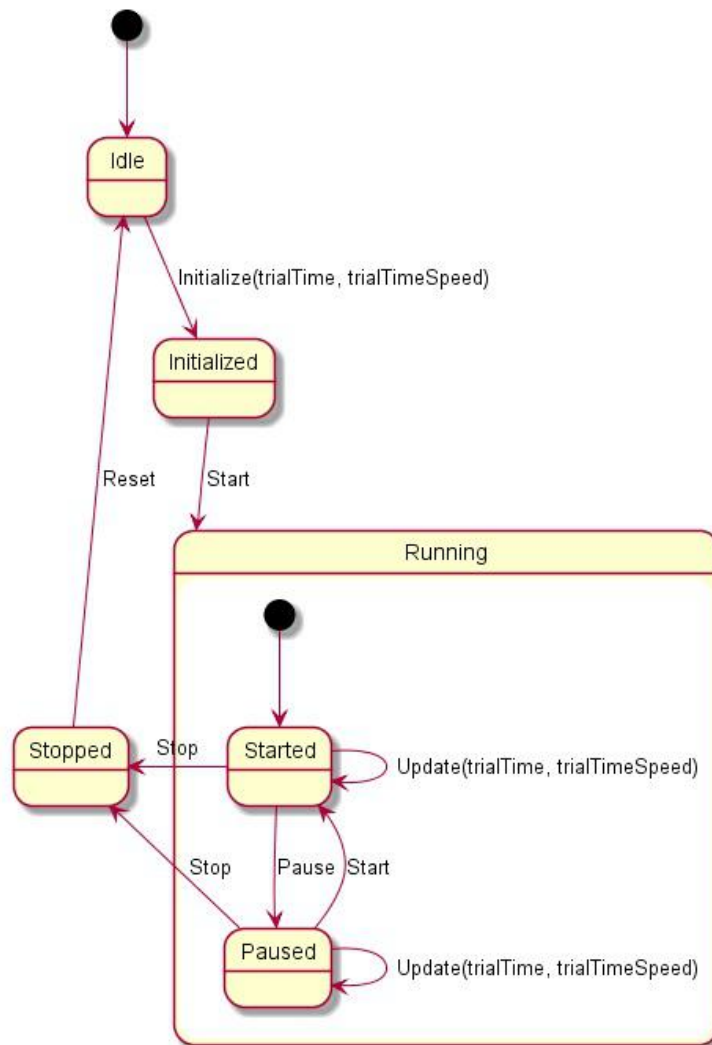


Figure 17. State diagram of the time service.

Even though you do not interface with the time messages, you still need to use the time interface when you need to send a message with a timestamp inside. This timestamp should use the current scenario time. Similarly, in case you display the 'actual' time in your user interface, please also use the current scenario time.

## System time versus scenario time

In Trials executed in the past, the operating system (OS) time was also adjusted to match the scenario time. The advantage was that if you would check the time in your status bar, it would display the current scenario time instead of the real time. Although this is straightforward to do, your OS does not like it, as it will generate files in the past or future, and may mess up your system. Especially when the scenario is paused. That's why the current Test-bed does not require you to synchronize your system time to the scenario time.



## A word about HLA and DIS

Within the Modelling & Simulation community, especially for military use, there are two simulation standards, [HLA](#) (High Level Architecture) and [DIS](#) (Distributed Interactive Simulation), which are the norm. The reasons why we did not use these standards, not even for the [CSS](#), are:

- They are used for connecting simulators to each other, not for connecting solutions to simulators nor solutions to other solutions.
- Their message format is fixed: if you want to send other information, you have to 're-purpose' existing fields, which is not considered a best practice. Also, they have no support for any [CM](#) standard.
- Both have a steep learning curve.
- [HLA](#) and [DIS](#) form a very small community, so it is difficult to hire people with this knowledge, and you typically have to train general software engineers by yourself. Second, it is difficult to find solutions for a particular problem on the Internet.
- [HLA](#) and [DIS](#) expect everyone to use Java or C++, and there is even less support for the 'newer' programming languages, like C#, JavaScript, Python, etc.
- [HLA](#) requires a run-time infrastructure, which is a kind of test-bed: there are two commercial providers that are rather expensive. Although there is one [open source version](#), it is feature incomplete and not well maintained. Although these versions should be interoperable, they are not, and they cannot be mixed.

That's why this Test-bed is using open source software, so it is easy to find:

- Open source tools to support it, or to connect to it, in many programming languages
- Answers to questions
- People that can use it
- A new schema representing your message
- And there is no financial hurdle preventing adoption

Even though the Test-bed does not use [HLA](#) or [DIS](#) internally, there are many simulators that provide a [HLA](#) or [DIS](#) export, and that can be useful for a [Trial](#). In those cases, a [HLA](#) or [DIS](#) simulation environment can be created, as is done normally, including a gateway service to bridge the gap with our test-bed: typically, such a gateway has an [HLA](#) connector to retrieve information from the [HLA/DIS](#) side, and a subset of the information is published in the [CSS](#). And vice versa. Even though this kind of integration is suboptimal, in practice, this is not really noticeable.



## 5. Test-bed design

The Test-bed is designed to fulfil the functional requirements, as described in D923.11, "Functional specification of the Test-bed". Clearly, different designs can be created that all fulfil these requirements, so this chapter provides a brief explanation of the major design decisions that underlie the current Test-bed's reference implementation. Its intended audience is core developers, who want to improve its functionality, or other backend developers, who want to create an alternative Test-bed that also satisfies these requirements.

### 5.1 Lessons learned from the Functional Specification

Part of the functional specification describes the [lessons learned](#) from D923.11. To summarize the most important technical lessons that have influenced the current design significantly, are:

1. The Test-bed should be open source.
2. The Test-bed should have a message-oriented architecture.
3. The Test-bed should use well-defined, easily accessible, syntactically correct messages, and close to common standards.
4. The Test-bed should be easily reproducible, and offer administrative as well as supporting tools and services.

### 5.2 Distributed message bus using Apache Kafka

Based on the functional requirements and lessons 1 and 2, and an analysis of many existing message-oriented systems, the Test-bed's backbone is built upon the distributed streaming service, [Apache Kafka](#). The main reasons to use Kafka are:

- Kafka allows for high performance sending and receiving of a very large number of messages
- Kafka allows for fast data replication and supports multiple receivers on the same message topic
- Kafka is a highly durable messaging system, persisting messages on the server or complete distributable file systems

- Kafka is a distributed system, making it scalable in the amount of message topics, senders and receivers.
- Kafka already has a large developer community, making it possible to easily use community-released tools to the current framework and assuring sustainability of Kafka.
- Kafka already has several security and message validation modules present, that will make it easier for simulators to safely connect to the [CSS](#).

Besides Apache Kafka, there are numerous popular open source messaging systems that were considered: [ActiveMQ](#), [RabbitMQ](#), and [ZeroMQ](#). The main reason for using Kafka, however, is its speed, low latency, and the fact that it is built from the ground up to be distributed. Especially for the simulators that are connected to the Test-bed, speed and low-latency are very important. And Kafka can easily process up to 100,000 messages per second, 10 times as much as the others. Its distributed nature allows to not only separate simulators and solutions, if required, but also supports having a reliable cross-site communication framework. Additionally, with its schema registry, it has excellent support for message validation out-of-the-box, which is detailed in the next section. The same applies to message persistency. Each message is immediately persisted to disk for a set time, which is easy for After-Action-Reviews, but also for clients that are not continuously online. In most messaging systems, when a consumer is briefly offline, the message is lost forever unless special care is taken to persist them.

### 5.3 Well-defined messages using Apache AVRO

Being able to communicate using well-defined messages is of primordial importance for any messaging system, and the Test-bed uses [Apache AVRO](#).

[AVRO](#) provides:

- Rich data structures.
- A compact, fast, binary data format.
- A container file, to store persistent data. For example, you could save all logged [AVRO](#) messages into a file, which also includes the schema file. This means that you will always be able to read the file at a later date, as it contains all the information to decode the messages.
- Remote procedure call (RPC).
- Simple integration with dynamic languages. Code generation is not required to read or write data files nor to use or implement RPC protocols. Code generation as an optional optimization, only worth implementing for statically typed languages.

In the Test-bed, each message consists of a key and a value, with:

- The **same** [AVRO](#)-encoded [key](#), based on the core attributes from the [EDXL DE](#) envelope (distributionID, senderID, dateTimeSent, dateTimeExpires, distributionStatus, and distributionKind). It can be used to support easy filtering and routing, and have a consistent message timestamp.
- A potentially different [AVRO](#)-encoded value, containing the actual message.

Each adapter verifies the key and value of each message before publishing (producing in Kafka terms) it. As Kafka limits you to have only one key/value schema pair per topic, there is no confusion when consuming a message about the schema's to use to decode a message's key and value.

Other evaluated candidates for defining message formats were [XML schema](#), Google's [Protobuf](#), and [HLA](#).

- XML schema is arguably the most popular format, especially when dealing with standards. However, it is very verbose, being text-based, which makes all messages very large. And it does not allow for schema migration, i.e. different versions of a message. This often leads to a 'creative re-use' of existing fields, which is common practice but not recommended.
- Protobuf is also a binary message format, like [AVRO](#), but also lacks schema migration.
- [HLA](#), yet another binary message format, is standard in the (Defence) simulation world, but of no use when defining messages in the [CIS](#) space.

## 5.4 Dealing with large messages

Kafka and [AVRO](#) are ideally suited for smaller-sized messages, i.e. typically not exceeding 1Mb compressed. However, a typical flooding file may well exceed 1Gb of data. The solution that currently is being designed is the following: in the Test-bed, there is one file hosting data services (e.g. FTP) for uploading large files. Each adapter, when confronted with a large message exceeding a threshold, uploads the data in the background to the file hosting data service, and in return, receives an obfuscated link. This link is subsequently shared via the Test-bed, and any consumer interested in the actual data can retrieve it from there.

A note about security: In principal, these files are openly available within the [Trial](#) environment (which may be on a closed network, of course). However, since the link is not easy recognizable, you will not be able to guess it. Basically, *security through obscurity*. This approach is similar to the one popular open file sharing services provide like [wettransfer.com](#), and should be sufficiently safe for current usage.

## 5.5 CIS and CSS adapters in several programming languages

On top of the communication framework, a set of guidelines need to be present that allows for external components and Test-bed components to communicate effectively. In this case, this set of guidelines is packed into a simple library called the adapter.

The adapter is the only form of connection between an application (solution, simulator, gateway, or otherwise) and the Kafka communication framework. There should be no other way to connect to it.

The adapter sets up and maintains the connection between application and the communication framework via several system message topics. The adapter deals with error handling and additional message validation, allowing the application to send and receive messages easily.

Since the Test-bed requires multiple solutions and simulators with different implementation frameworks to be connected to the CIS/CSS, multiple adapters are created. Currently, an adapter for [Java](#), [JavaScript/TypeScript](#) and [C#](#) is created and can be found in the DRIVER+ GitHub (<https://github.com/DRIVER-EU>). A [REST](#) endpoint is also present for web services to communicate with the [CSS](#) in a similar fashion. All adapters should have similarly behaving functionality with a clear and understandable API.

Adapters extend regular Kafka connectors with the following information, each of which is displayed in the Test-bed's admin tool:

- *Heartbeat signals*: Before you can start a [Trial](#), every solution, simulator and tool needs to be up-and-running. Therefore every adapter transmits a heartbeat signal every 5 seconds to inform others it is online.
- *Logging*: Besides being online, it is also important to know that each connected service is running as expected, so each adapter offers the option to log warnings/errors to the Test-bed as well.
- *Configuration options*: The adapter can inform others to what topics it subscribes and publishes. In addition, this can be configured too externally. For example, the admin tool can configure the (potentially secret) topics an adapter must listen too.
- *Time*: A [Trial](#) scenario typically will not run at real-time, so the adapter needs to share the fictive simulation time. In addition, it shares the simulation speed, as we may be running slower or faster than real-time, as well as the simulation state.

# Abbreviations

Abbreviation	Definition
AAR	After Action Review
AVRO	Open data serialization system supported by the Apache Organisation. <a href="http://avro.apache.org">avro.apache.org</a>
C++	[C plus plus] Programming Language
C#	[C-sharp] Programming Language
CAP	Common Alerting Protocol, standard data format for exchanging public warnings
CIS	Common Information Space for exchanging crisis management messages
CM	Crisis Management, see also <a href="#">Annex 1</a>
CSS	Common Simulation Space for exchanging simulation data
COP	Common Operational Picture tools for creating a shared situational awareness
CoPCM	Community of Practice in Crisis Management
DIS	Distributed Interactive Simulation, a simulation standard <a href="http://en.wikipedia.org/wiki/Distributed_Interactive_Simulation">en.wikipedia.org/wiki/Distributed_Interactive_Simulation</a>
Docker	Container environment to enable independence between applications and infrastructure
EDXL	Emergency Data Exchange Language, set of message standards defined by OASIS to share emergency information, <a href="http://docs.oasis-open.org/emergency">docs.oasis-open.org/emergency</a>
EMSI	Emergency Management Shared Information, message format for the exchange of emergency information
GitBook	Open Source service for creating online books
GitHub	Repository for managing DRIVER+'s software code
GT	Guidance Tool
GUI	Graphical User Interface
HLA	High Level Architecture, a simulation standard <a href="http://en.wikipedia.org/wiki/High-level_architecture">en.wikipedia.org/wiki/High-level_architecture</a>
Java	Programming Language
JavaScript	Programming Language

Kafka	An open source distributed streaming platform supported by the Apache Organisation that is used as the basis to exchange information between simulators and solutions
KPI	Key Performance Indicator
MBtiles	Single file database format to store images of a map
MGT	Management
PoS	Portfolio of Solutions, a DRIVER+ content management system to maintain a list of <a href="#">CM</a> solutions, their application, strength and weaknesses, and user reviews.
Python	Programming Language
<a href="#">REST</a>	Representational State Transfer (common interface allowing you to read and write data from a service)
RTI	Run-time infrastructure, a kind of <a href="#">HLA</a> -based Test-bed
<a href="#">SA</a>	Situational awareness, basically do you know on the map where your people and other resources are, as well as all relevant crisis management related incidents and activities.
SUMO	Simulation of Urban Mobility, a traffic and pedestrian simulator
<a href="#">TGM</a>	<a href="#">Trial</a> Guidance Methodology, as defined in deliverable D922.21, " <a href="#">Trial</a> guidance methodology and guidance tool specifications (version 1)".
TypeScript	Programming Language
WFS	Web Feature Service (serving a map layer as vectors)
WMS	Web Mapping Service (serving a map layer as picture)
XACML	eXtended Access Control Markup Language, a standard for describing security permissions to resources
XML	eXtended Markup Language, a textual representation of a message that is easily readable by computers

## Annex 1: DRIVER+ Terminology

Terminology	Definition for DRIVER+	Source
Crisis management	Holistic management process that identifies potential impacts that threaten an organization and provides a framework for building resilience, with the capability for an effective response that safeguards the interests of the organization's key interested parties, reputation, brand and value creating activities, as well as effectively restoring operational capabilities. Note to entry: Crisis management also involves the management of preparedness, mitigation, response, and continuity or recovery in the event of an incident, as well as management of the overall programme through training, rehearsals and reviews to ensure the preparedness, response and continuity plans stay current and up-to-date.	ISO22300 (DRAFT 2017) 8
Evaluation	Process of estimating the effectiveness (3.1.3.03), efficiency (3.1.3.04), utility and relevance of a service (3.1.1.59) or facility "ISO 5127:2017(en)	
Exercise	Process (3.180) to train for, assess, practise and improve performance (3.167) in an organization (3.158) Note 1 to entry: Exercises can be used for validating policies, plans, procedures (3.179), training (3.265), equipment, and inter-organizational agreements; clarifying and training personnel (3.169) in roles and responsibilities; improving inter-organizational coordination (3.52) and communications; identifying gaps in resources (3.193); improving individual performance and identifying opportunities for improvement; and a controlled opportunity to practise improvisation. Note 2 to entry: See also test (3.257).	ISO22300 (DRAFT 2017) 11
Experiment	Purposive investigation of a system through selective adjustment of controllable conditions and allocation of resources	ISO/TR 13195:2015(en) Selected illustrations of response surface method — Central composite design, 2.1
Trial Guidance Methodology	A structured approach from designing a Trial to evaluating the outcomes and identifying lessons	DoW



(TGM)	learned	
Guidance Tool	A software tool that guides <a href="#">Trial</a> design, execution and evaluation in a step-by-step way including as much of the necessary information as possible in form of data or references to the portfolio of solutions	DoW
Interoperability	The ability of diverse systems and organisations to work together, i.e. to interoperate.	ISO 22397
Legacy systems	(Crisis management) system currently in operational use.	Initial DRIVER definition
Observer	<a href="#">Exercise</a> participant who watches selected segments as they unfold while remaining separate from role player activities [DRAFT 22300: 2017-- observer participant (3.163) who witnesses the <a href="#">exercise</a> (3.83) while remaining separate from <a href="#">exercise</a> activities Note 1 to entry: Observers may be part of the evaluation (3.81) process (3.180).]	ISO 22300:2012(en) Societal security — Terminology, 2.4.5 [addition in DRAFT 2017]
Portfolio of Solutions (PoS)	A database driven web site that documents the available Crisis Management solutions. The PoS includes information on the experiences with a solution (i.e. results and outcomes of Trials), the needs it addresses, the type of practitioner organisations that have used it, the regulatory conditions that apply, societal impact consideration, a glossary, and the design of the Trials.	DoW

# List of Figures

1. [PTME paradigm applied to DRIVER+.](#)
2. [CIS and CSS.](#)
3. [Test-bed reference implementation.](#)
4. [Test-bed environment.](#)
5. [Scope of the test-bed.](#)
6. [Explanation of the Test-bed components.](#)
7. [Admin tool.](#)
8. [JEMM exercise script example.](#)
9. [Exonaut timeline example.](#)
10. [Observer Support Tool: Left, an overview of available observation templates. Right, one of the observation templates is selected.](#)
11. [Test-bed composer's home page.](#)
12. [Test-bed composer: Selecting a solution.](#)
13. [Test-bed composer: Downloading the docker-compose.yml file.](#)
14. [Screenshot of Landoop's Kafka topics UI, which is part of our test-bed.](#)
15. [Screenshot of Landoop's AVRO schema registry, which is part of our test-bed.](#)
16. [Screenshot displaying the Kafka-replay service's Swagger interface.](#)
17. [State diagram of the time service.](#)