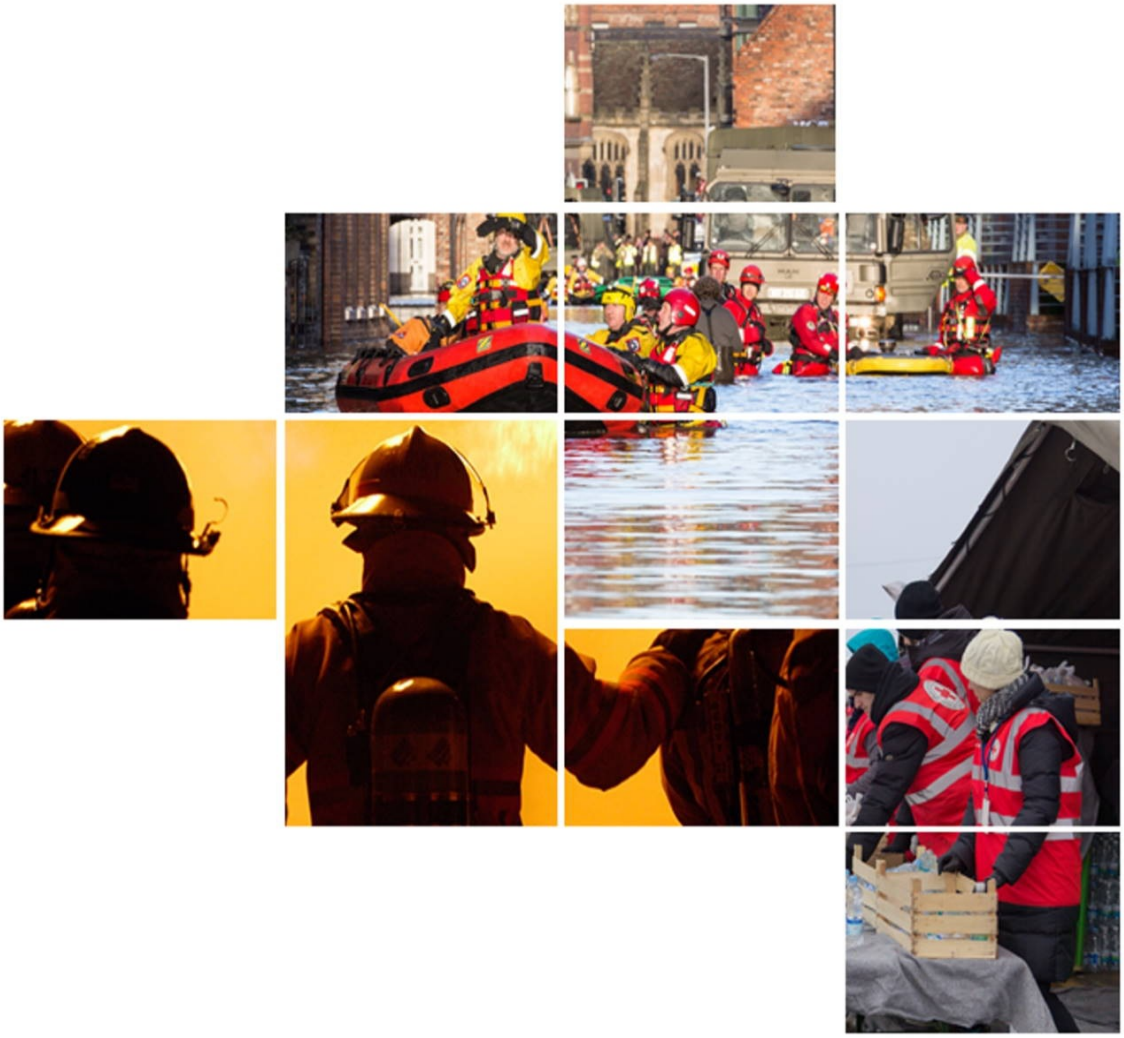
driver+

Driving Innovation in Crisis Management for European Resilience



D923.23 THIRD RELEASE OF THE TEST-BED REFERENCE IMPLEMENTATION

SP92 - TEST-BED

SEPTEMBER 2019 (M65)

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

Project information

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

Document information

Document Status:	Final
Document Title:	D923.23 Test-bed reference implementation v3
Document Nature:	Report (R)
Dissemination Level:	Public (P)
Due Date:	30/09/2019
Submission Date:	30/09/2019
Sub-Project (SP):	SP92 - Test-bed
Work Package (WP):	WP923 – Test-bed infrastructure
Document Leader:	TNO
File Name:	test-bed-design.pdf, also available as epub or mobi eBook. The latest version can be found online at https://driver-eu.github.io/test-bed-design

Revision table

Issue	Date	Comment	Author
V1.1	16/08/2018	Updating documentation for the second release	Erik Vullings, TNO
V1.2	15/11/2018	Review process started	Erik Vullings, TNO
V1.3	25/11/2018	Cloud and reverse proxy section added	Pieter Hameete, TNO
V1.4	26/11/2018	Reviews from Héctor Naranjo (GMV), Laurent Dubost & Antoine Léger (TCS), Michael Middelhoff (WWU), and Chiara Fonio (JRC) processed	Erik Vullings, TNO
V1.5	27/11/2018	Document review	Angela Schmitt, DLR
V1.6	28/11/2018	Preparation of final version	Erik Vullings, TNO
V1.7	29/11/2018	Final check and approval for submission	Tim Stelkens-Kobsch, DLR, Quality Manager
V1.8	29/11/2018	Final check and approval for submission	Marijn Rijken, TNO, Project Director
V2.0	30/11/2018	Submission to the EC	Francisco Gala, ATOS
V2.1	20/08/2019	Updated and included latest changes	Erik Vullings, TNO
V2.2	06/09/2019	Reviews from Héctor Naranjo (GMV), Laurent Dubost & Antoine Léger (TCS), Dennis Horstkemper (WWU) processed	Erik Vullings, TNO
V2.3	13/09/2019	Final check and approval for submission	Tim Stelkens-Kobsch, DLR, Quality Manager
V2.4	20/09/2019	Final check and approval for submission	Marijn Rijken, TNO, Project Director
V3.0	30/09/2019	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 or visit the animation online at https://vimeo.com/299680658). It provides an open source technical backbone to perform Trials or exercises in a methodical and structured way by offering Crisis Management practitioners a suite of free and open source *software* tools. This document discusses the Test-bed's usage and design from the perspective of a practitioner. The later chapters, however, are aimed at developers and system administrators, and therefore assume that the reader has at least some technical background. In the 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 deliverable D923.21.

> The current version of this document discusses the **third version** of the Test-bed reference implementation. Therefore, all returning readers are advised to read the change history carefully, as many parts of this document did not change.

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 (TGM) 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.

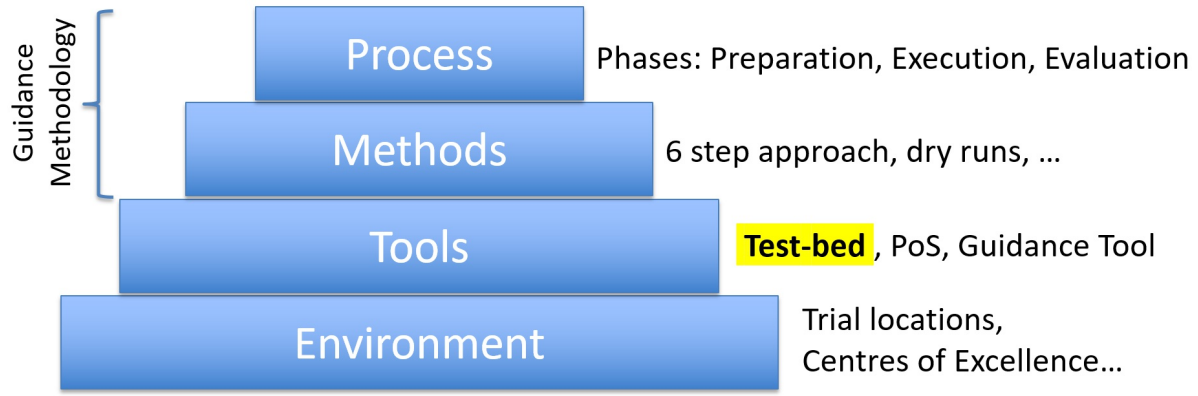


Figure 1. Process-Methods-Tools-Environment (PMTE) paradigm, where the TGM prescribes the Process and Methodology, supported by the Test-bed technical infrastructure (Tools)..

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 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 implementation 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/NodeJS, Python and REST are available.

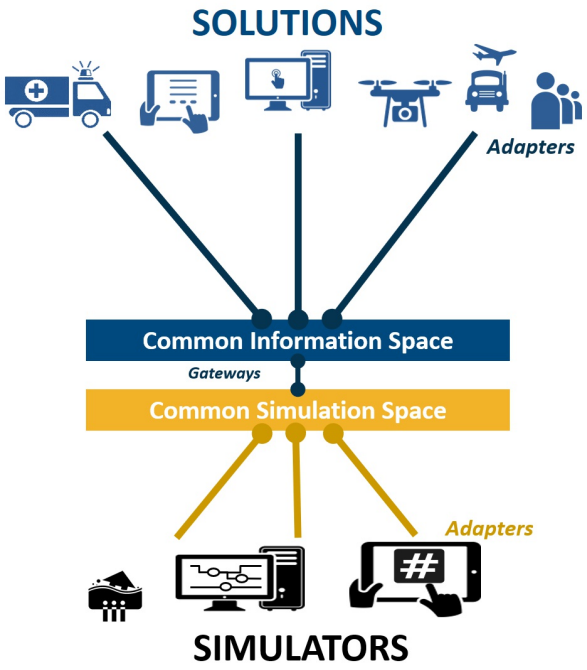


Figure 2. The Common Information and Simulation Space allow the exchange of well-structured, informative messages.

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.

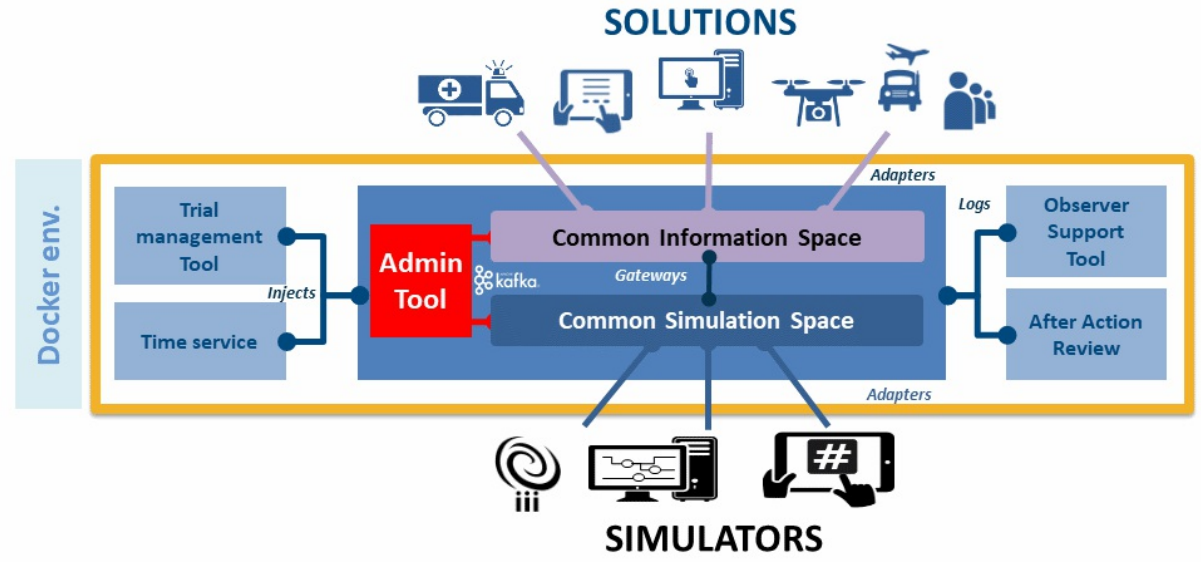


Figure 3. Test-bed reference implementation: the components inside the yellow rectangle are being developed as open source tools in the DRIVER+ project.

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. Trial-Management-Tool (TMT) and After-Action-Review (AAR) tool are not included and other components in first prototype quality). Version 2 was used in Trial 4 with a first version of the TMT and AAR, and the current version 3, as described in this document, will be used in Trial 3 and the Final Demo.

All software is already available open-source (under a liberal MIT license) on github.com/DRIVER-EU. The subsequent updated Test-bed reference implementations are documented in D923.21, D923.22, and D923.23 (this document) - Reference Implementations of the Test-bed (versions 1/2/3 resp.).

Besides the officially released software components, as described in the DRIVER+ DoW and in this document, the DRIVER-EU space on GitHub also contains many other useful services that were developed in order to support a Trial's specific needs. For example, Trial 4 required a deep integration between the Test-bed and the Dutch Crisis Management System (LCMS). This has led to a connector that extracts text and map overlays and vice versa. This connector is also being used in an upcoming table-top exercise (organised by the EU project IN-PREP) that is situated in The Netherlands, and will be supported in the future by the Dutch national CM CoE, IFV. Or on the simulator side, a connector with the open source SUMO traffic simulator was created, so future trials or exercises can operate against a realistic traffic background, or use it for dispatching virtual ambulances. Other examples include the lesson's learned framework website, an icon and large file service, a geo-server that triggers when people or entities are at a certain location, an email gateway to add email services to the Test-bed, a Twitter service that sends messages to (preferably a private) twitter account, to name but a few. This document, however, focuses on the main deliverables needed to setup a Trial.

This document is a living document, and the latest version is available online at https://driver-eu.github.io/test-bed-design or can be downloaded as PDF, epub or mobi eBook [1].

> [1]. 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. Their aim is to be more efficient and effective, save more lives, against fewer costs. Some of the ways to achieve this are to improve existing CM procedures, like media communication. Other possibilities are to introduce new software applications or hardware to remedy current shortcomings, such as introducing a Common Operational Picture application when the situational awareness is insufficient. But how does one really know that such a solution improves things? By trying them out in a Trial, a well-controlled experiment carried out under observation.

The DRIVER+ project aims at helping CM practitioners by providing them with the 'tools' to properly evaluate new solutions during a Trail:

- The Portfolio of Solutions (PoS), an online web application where the CM practitioner can find out about existing solutions, and user experiences.
- A Lessons Learned Framework (LLF), an online web application to share experiences among practitioners.
- The Trial Guidance Methodology (TGM, deliverable D922.21 - Trial guidance methodology and guidance tool specifications (version 1)) and Trial Guidance Tool (TGT), a step-by-step approach that guides CM practitioners through all the steps needed to prepare and run a Trial.
- The Test-bed infrastructure, a suite of software tools and services that support practitioners preparing and executing a Trial or training exercise

This document is all about the latter, the Test-bed infrastructure, but since it strongly relates to the TGM, a brief summary of the TGM is presented first.

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 3). 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.

1.1 Aim of this Document

This document aims at providing the CM practitioners with a clear understanding of what the Test-bed is, and what it can do for his or her organisation. Especially chapter 1 and 2 will be of interest to them, and for the more technical oriented CM practitioner, also chapter 3. Chapter 3 and 4 are concerned with the technical setup, and are directed at developers or system administrators that need to connect new solutions or setup and maintain a Test-bed.

1.2 Scope of the document

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 3): 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.

1.3 Organisation of the Document

This is a live document, and the latest version can always be found online at https://driver-eu.github.io/test-bed-design. It is organised as follows:

For **reviewers**, please refer to Annex 2 Change history to see what has changed since the previous version. In order to quickly understand the Test-bed, have a look at the 5 minutes animation at https://vimeo.com/299680658.

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.

The Test-bed description provides a general overview of the Test-bed reference implementation and its building blocks. It is a more technical chapter aimed at Test-bed users with a (limited) technical background.

In the chapter, Test-bed for developers and sysops, the technical side of the Test-bed is explained in more detail. Specifically: how to manage a Test-bed as a sysop, or how to connect a new Solution or simulator as a developer. To learn more about the design principles underlying the Test-bed, the reader is referred to Annex 3 Test-bed design.

1.4 What's new

Version 1 (2018-03-31)

This is deliverable D923.21, describing the first version of the Test-bed reference implementation. It matches the release version of the software applications as available on GitHub: 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.

Version 2 (2018-11-30)

This is deliverable D923.22, describing the second version of the Test-bed reference implementation v2. All changes with respect to the previous version are documented in Annex 2 Change history.

Version 3 (2019-08-31)

This is deliverable D923.23, describing the third and final version of the Test-bed reference implementation v3. All changes with respect to the previous version are documented in Annex 2 Change history.

2. A Test-bed for Crisis Management Practitioners

This Chapter discusses the **future** Test-bed functionality that could benefit a practitioner, so although 95% of the functionality can already be realized today, not everything is in place yet and we will continue improving it until February 2020.

Trial owners will interact with the Test-bed when they want to Trial one or more solutions (see Figure 4). 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 a Solution has useful features, is user-friendly, looks good, and does not 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.

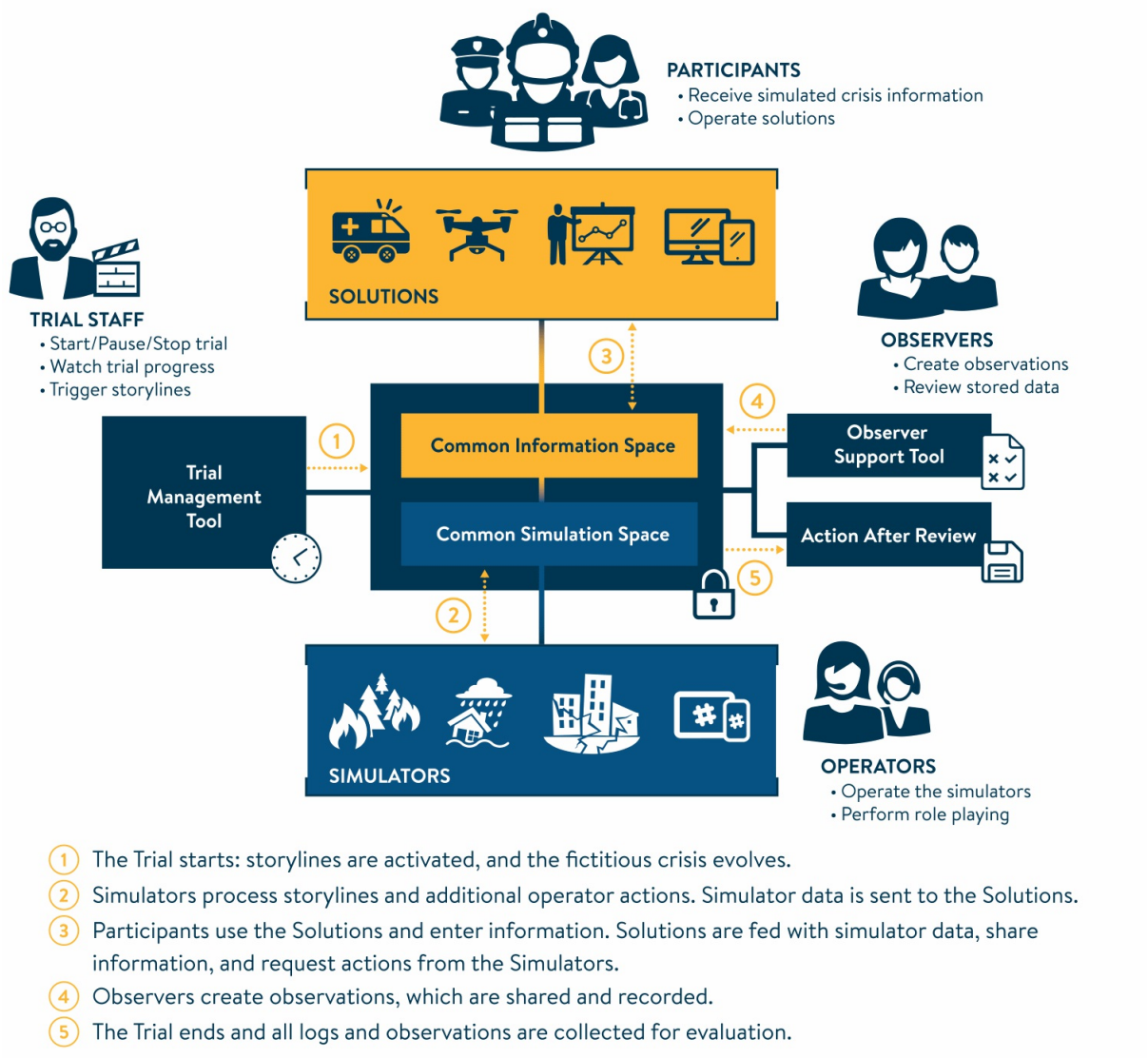


Figure 4. High-level overview of the technical infrastructure needed for a trial.

> For example, if one would test a COP tool as is, it would probably not have any map layers, basically showing an empty map. Using a test-scenario, the COP tool can be tested in the context of a simple incident, perhaps a wildfire, displaying the actual locations of fire trucks and firemen, and map layers showing population information, locations of critical infrastructure, fire

> hydrants, 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.

> The Test-bed is, however, also suitable for supporting table top exercises or field exercises, since it contains all the components that are required for it: a common information and simulation space to exchange information between tools, the Trial-Management-Tool for creating and running scenario's, and review and observer support tools. In the scope of this project and document, however, the focus is on Trials that want to evaluate CM solutions.

For simple as well as complex cases, the Test-bed supports practitioners by providing an environment in which they can easily Trial new solutions and run exercises:

- **Preparation support:** When preparing a Trial, Trial staff can use the Trial-Management-Tool tool to create one or more scenarios for the Trial. In addition, they can already document the observations they wish to see using the Observer Support Tool. As part of the preparation, the Test-bed also offers the Admin tool to manage the Trial's technical setup, and:
  - **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. As most CM simulators cannot simulate realistically all types of incident and reactions, the Test-bed supports multiple simulators so they can collaborate and create the desired fictitious incident.
  - **Solution support:** The new and shiny COP tool or other solution that is being evaluated never acts standalone in a void. Normally, solutions need to exchange information with other (legacy) solutions too, which is also supported by the Test-bed.
  - **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, developers and system operators can use the Admin tool.
- **Execution support:** Scenarios created during the preparation phase can be 'executed', or played, in order to control the Trial. And the scenario time is controllable too.
- **Evaluation support:** As we are testing and evaluating new solutions, the Test-bed provides the After-Action-Review tool for capturing and reviewing sent messages and the Observer Support tool to capture observations. In addition, the exchanged information can be logged and processed for further evaluation after the 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. Please also have a look at the animation, which is available online at https://vimeo.com/299680658.

2.1 Use Case: Evaluating a solution standalone

Monica has heard about an interesting COP solution via the DRIVER+ Portfolio of Solutions. She considers using it to address her problems in fighting fires. After defining the gaps and specifying the requirements, according to the TGM, she continues in an iterative way:

1. Monica, supported by an IT professional, visits the Test-bed's website (see Figure 5). *It allows her, or anyone else, to specify the technical components that should make up the Test-bed. Alternatively, visit the Test-bed's website and select one of the available (Docker-based) Test-bed definitions from the* `docker` *folder. Download the folder and you can run them instantly (if Docker is installed on your PC).*
2. She opens the solution's tab and sees that the COP tool is available for evaluation. *In the future, available solutions could be linked to the Portfolio of Solutions.*
3. She selects it in the menu (see Figure 6).
4. 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.*

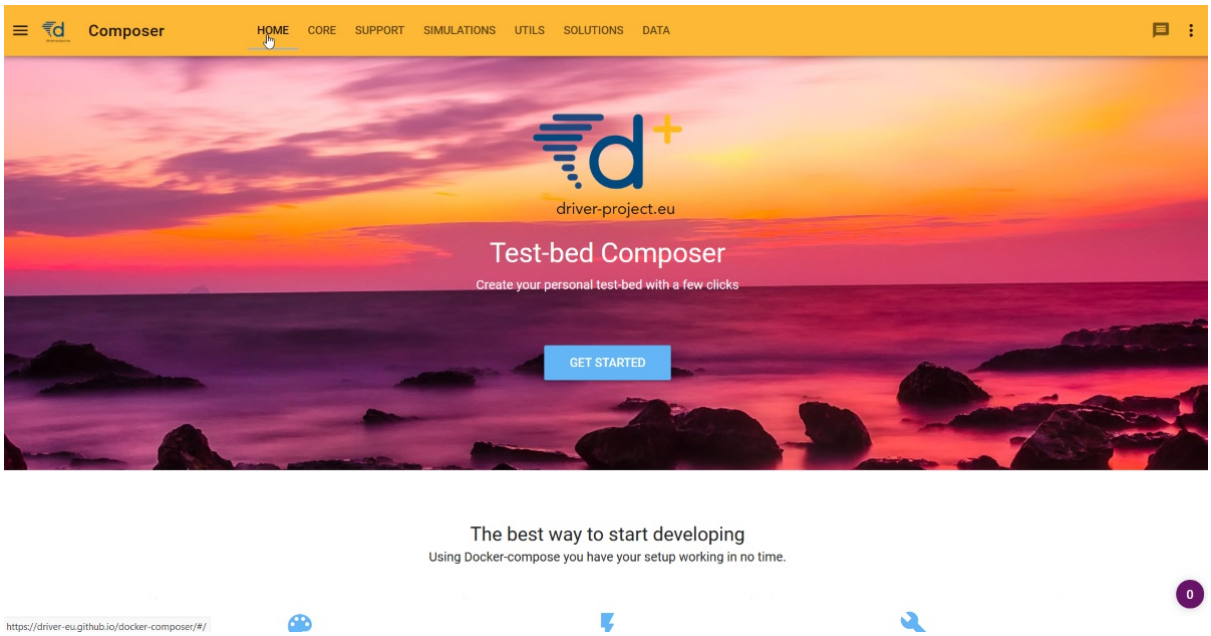


Figure 5. Test-bed composer's home page.

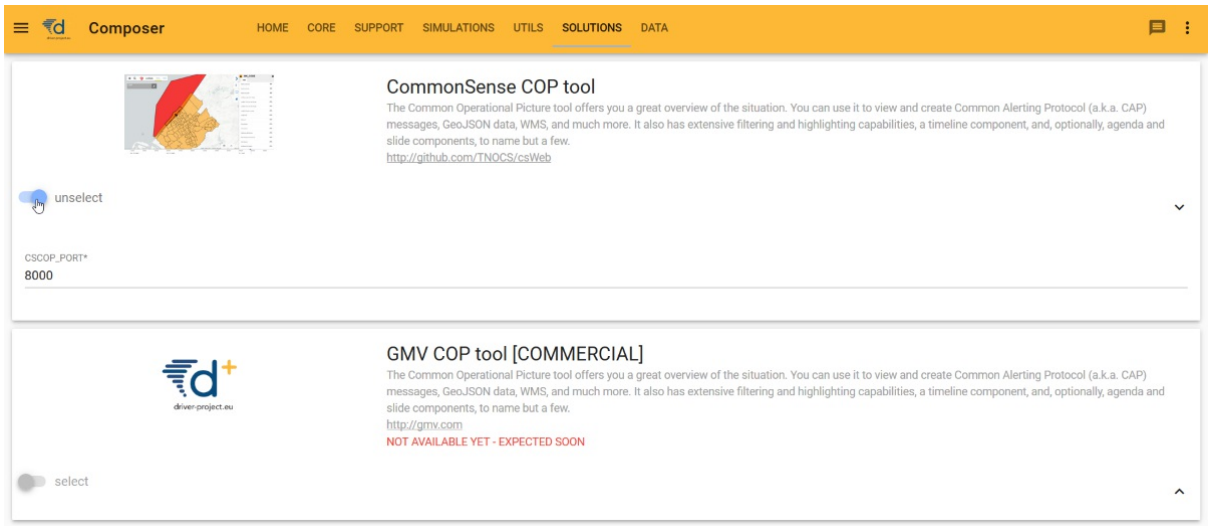


Figure 6. Test-bed composer: Selecting a solution.

From here on forwards, two alternatives are presented.

Alternative 1: Cloud scenario

1. As she currently is not 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 brief moment, 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 COP 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.

> The above scenario still represents the ideal situation. Currently, however, the Cloud scenario is implemented as follows: From the Test-bed website at https://github.io/DRIVER-EU/test-bed, in the `docker` folder, a Docker file is selected and uploaded to a Cloud service. Although many cloud services can be used, the current version is based on two open source solutions, `Docker swarm` and `Portainer` as GUI. The responsible person only needs to make sure that there are no conflicts with other running versions (especially checking for port conflicts).

Alternative 2: Local scenario

1. As she currently is not 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 7).
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.

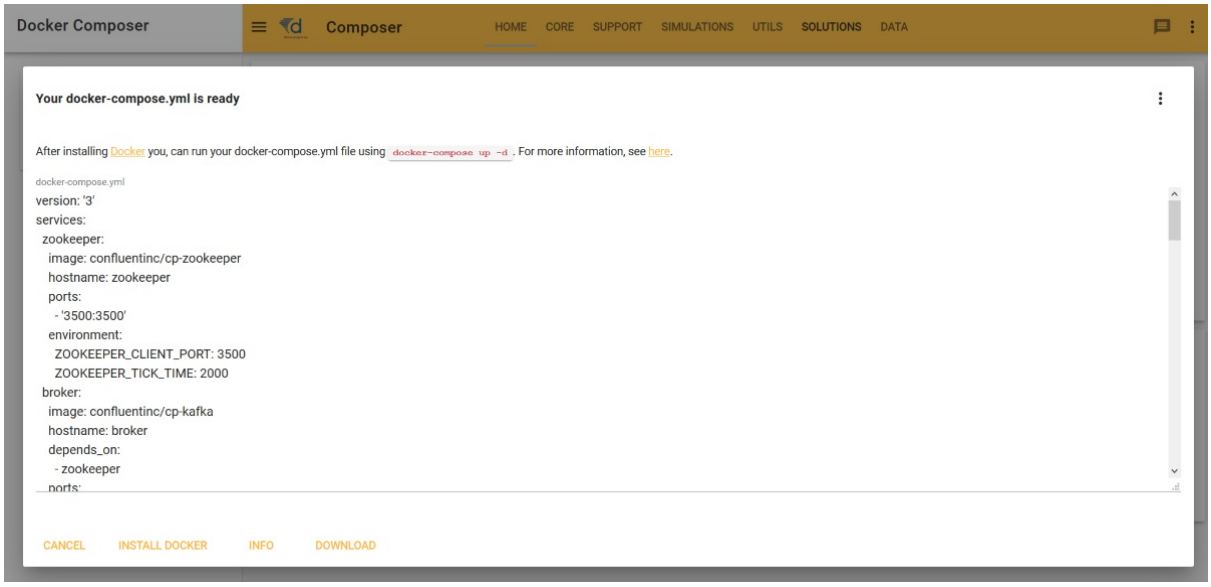


Figure 7. Test-bed composer: Downloading the docker-compose.yml file.

> In practice, during all Trials, it turned out to be easier to just take one of the existing Test-bed descriptions from the Test-bed website at https://github.io/DRIVER-EU/test-bed and adapt it manually as needed: comment out those sections that you don't need, change a port number to be in an available range, etc. An example of such a so-called `docker-compose.yml` file is shown in Annex 4 Docker Test-bed example.

2.2 Use Case: Assessing solutions using a Trial

In order to run a Trial, a similar process as described above could be followed. The main difference is that more services and solutions will be added, and more people are involved. In addition, the Test-bed is needed already well-before the final Trial date, since:

- The scenario is 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.

In this case, Monica is supported by the local CM Centre Of Excellence, which could also be offered via a consultant, a simulator owner, or otherwise, which helps her in setting up the required infrastructure.

1. She contacts one of the local CM Centres Of Excellence, and request a Test-bed in the cloud. It is created for her.
2. She contacts the solution provider, and informs him of her Test-bed. This solution owner can contact the CM Centre Of Excellence, and with help of its IT staff, are able to connect to Monica's Test-bed, so Monica can have a first look. This connection to the Test-bed is needed, so the Solution can receive information about a fictitious incident, or from other solutions in a more complex Trial.
3. As part of the online Test-bed, she now has access to the Trial-Management-Tool Tool (TMT, see Chapter 3), formerly known as the Scenario Editor, which allows her or one of her colleagues, to create the outlines of several wildfire scenarios by recording the main flow of events during an incident. The scenario is key in that it translates specific objectives to a storyline so the solutions or people can be put to the test:
4. As the TMT does not create the actual incident itself, she realizes that she needs some simulators to simulate a wildfire, traffic and evacuating civilians. In DRIVER+'s Portfolio of Solutions, she finds several simulators that are compatible with the Test-bed and contacts their owners. The simulation providers, like the solutions provider, are able to connect to Monica's Test-bed too. Alternatively, it may be offered by the CM Centre Of Excellence too. Using the Test-bed connection, the simulators can send incident information to the connected Solution, which it can display. In addition, simulators can share information too, so simulated people do not walk happily through the fire!
5. Monica asks one of her colleagues to detail the questions for the observers, which he can document in the Observer Support Tool (OST, see Chapter 3).
6. During a Trial, Monica can start the scenario. The Trial-Management-Tool Tool instructs everyone when to perform their planned act. For example, the TMT instructs the Observer Support Tool to send the first set of questions to the observers. Monica can also control the scenario time, for example by freezing the time during a break. In case the users of a Solution missed an important test moment, she can also invoke optional storylines or acts, so the Solution and its users get another chance to prove itself.

3. 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 3) and is aimed at a fairly technical audience. A shorter version was recently presented at ITEC2019, "An interoperability Framework for Trials and Exercises", by Hendriks, Vullings, Van Campen2, and Hameete.

Although most trialled DRIVER+ solutions are software systems, the Test-bed also offers excellent support for non-software systems, such as a new process or procedure.

- The Test-bed's technical infrastructure can be used to put a new process in a relevant crisis-management context. For example, the open source SUMO tool could be used to simulate traffic, and the open source csCOP tool could be used to offer the participants a realistic common operational picture during their Trial. There are also organisations that provide commercial support to provide the Test-bed with realistic incident visualisations.
- Within the Test-bed, there are three tools that can also be of direct value for trialling new procedures: the Trial-Management-Tool (see Section 3.3), the Observer-Support-Tool (Section 3.4), and the After-Action-Review tool (Section 3.4). The first to provide stimuli to the procedure, the second to capture all observations, and the last to review the whole trial session.

3.1 Core

The Test-bed must support the exchange of information between 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. These messages can have different effects, for example:

- Display a message to an end-user such as an email or an alert on a map.
- Track the location of your own people on a map.
- Share the current situational status between CM solutions.
- Dispatch an ambulance to the incident location.
- Receive an update of the wildfire location from the drone system.
- Trigger a flooding simulator to send a new flood map.
- Make part of the road network inaccessible for traffic.
- Sharing an observation made by an observer, which can be used as input to the CM scenario.
- Send a message to a role-player, who subsequently makes a call as a concerned citizen to the crisis communication centre.
- Mark the begin or end of a CM phase, useful for after action analysis.

Although already many messages are implemented, it is easy to add more, based on the needs of the actual Trial.

Each message is published in a so-called *topic*, where each topic can only be used for one type of message. To receive these messages, all an application needs to do is to subscribe to the topic of interest, using an adapter.

Adapters

Adapters are used to connect solutions and simulators to the Test-bed in order to exchange information. As the Test-bed is based on Apache Kafka, which is used worldwide, there are connectors for most programming languages, so software applications can easily connect to it. While connected to the Test-bed, an application can send and/or receive messages. When you want to *receive* a message, you subscribe to a topic of interest: thereafter, you get all the messages that are published until you end your subscription. Optionally, you can even receive messages that were published in the past, or while you were offline, as Kafka logs all messages for a configurable time. This is especially useful for mobile services that are not always connected to a network. To *publish* a message, you just need to send it to a topic of interest and every interested application that has subscribed to this topic receives it instantly.

For example, to publish a CAP (Common Alerting Protocol) message to all interested parties, you use an adapter 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, however, are lacking certain features that are useful in a Test-bed environment, so several existing connectors have been extended. These extended connectors are called **adapters**, which 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. This is monitored in the Admin Tool.
- *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. The logs are also displayed in the Admin Tool.
- *Configuration options:* The adapter can be instructed via the Admin tool to subscribe and publish to certain topics. Alternatively, in development mode, the adapters are free to connect to any topic.
- *Trial 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 (paused/playing/stopped).
- *Large files:* Although most messages that are shared between applications are small (< 1Mb), some are not. For example, to describe the flooding in an area, or when capturing drone imagery, file sizes may easily exceed 1Gb. For that reason, the Test-bed runs a service to share large files, and adapters facilitate using this service: the large file is uploaded to the service, and the adapter either sends a message when ready, or informs the application. Other applications can subsequently use the returned (URL) address to access the file.
- *Security:* By default, all topics are accessible to all connected adapters, which is fine in most cases. However, sometimes a less liberal access is required, e.g. when sharing the locations of critical infrastructure or network configurations. All adapters, therefore, include a security component which prevents unauthorized users from accessing specified topics.

The Test-bed currently maintains the following adapters: Java, C#, JavaScript/TypeScript/Node.js, Python, and a REST adapter, which allows any application to publish and receive messages using basic internet commands.

AVRO Message types

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 (for a brief example, see below). All Test-bed schema's are shared publicly in the DRIVER+ AVRO schema repository, which also contains more relevant schema's for CAP or EMSI messages. An excerpt of a CAP schema is shown below.

```
// Excerpt of a Common Allerting Protocol schema. The complete schema can be found at
// https://github.com/DRIVER-EU/avro-schemas/blob/master/standard/cap/standard_cap-value.avsc.
{
  "name": "eu.driver.model.cap.Alert",
  "namespace": "eu.driver.model.cap",
  "doc": "CAP Alert Message (version 1.2)",
  "type": "record",
  "fields": [
    {
      "name": "identifier",
      "type": "string"
    },
    {
      "name": "sender",
      "type": "string"
    },
    {
      "name": "sent",
      "type": "string",
```

```
      "doc": "TODO xs:dateTime Used pattern"
    },
    {
      "name": "status",
      "type": {
        "name": "Status",
        "namespace": "eu.driver.model.cap",
        "type": "enum",
        "symbols": [ "Actual", "Exercise", "System", "Test", "Draft" ]
      }
    }
    // ...
  ]
}
```

Dealing with standards

The Test-bed enforces AVRO schemas, as it:

- Is able to process and distinguish between various types of Messages formats such as CAP, EMSI.
- Can enforce that only valid messages are shared.
- Is concise.

In the CM domain, several standards exists, such as CAP, EDXL or EMSI. They are represented using XML, a textual representation of a message that is easily readable by computers, and which are formalized using XML schemas. Although the Test-bed could have used XML messages directly, there is currently no way to validate this inside Kafka, i.e. a topic that should only deal with CAP messages would still accept EMSI messages, since they are both XML, which would lead to exceptions in the topic subscribers. Even trickier are invalid CAP messages that do not respect the XML schema, or use a different version of the schema. Also XML messages are much more verbose than the binary AVRO messages.

For all of these reasons, the Test-bed enforces all topics to use AVRO schemas, and the XML schemas are converted to AVRO schemas. But this comes at a small price, as this conversion is not perfect since XML schemas are slightly more expressive. For example, an XML schema can enforce a string to adhere to a specific pattern, such as a phone number, which is not possible in AVRO:

```
<xsd:simpleType name="phoneType">
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="[0-9]{3}-[0-9]{7}"/>
  </xsd:restriction>
</xsd:simpleType>
```

However, since most solutions dealing with CM standards should already have formatted these messages correctly, this should not lead to many problems. Within the DRIVER+ Trials, at least, it worked well.

Finally, another advantage of using AVRO schemas is that they are shared in the Test-bed via the schema registry

CIS and CSS

The Test-bed distinguishes between a Common Information Space (CIS) and a Common Simulation Space (CSS). The CIS is where *solutions* exchange information, typically based on current emergency standards such as the Common Alerting Protocol (CAP), or Emergency Data Exchange Language (EDXL). The CSS is for *simulators* to exchange information. Typically, the CIS will exchange far fewer messages during a Trial, and time synchronisation is simple. In the CSS, however, many more messages are generated, e.g. the location of all vehicles may be updated every second. Simulators often need to be in tight synchronization with each other, e.g. when a flooding simulator floods an area, the traffic simulator should divert its traffic, and in the crowd simulator, people may be drowning.

For simple Trials, the CSS and CIS will run in the same Test-bed. In case performance suffers, it may be necessary to split the CSS and CIS over two Test-beds that are interconnected, a feature that Apache Kafka supports out-of-the-box.

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 when using well defined messages based on Apache AVRO, it is certain that not all solutions and simulators speak each other's 'language'. As in Europe, where not everyone is speaking Esperanto or English, 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 in the CSS, to a COP tool message in the CIS that only contains the location of its own resources.
- An EDXL Resource Management request from a COP tool in the CIS to a simulator message in the CSS, which in turn sends out an ambulance to the required location.
- A flood map as generated by the flooding simulator containing the actually flooded area, to a flood map based on the available sensor information containing the realistically known flooded area.

In order to facilitate Solutions to obtain the 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 (see Figure 8). These translator applications form the bridge between the simulated *truth* in the CSS, where everything is known, and the *perceived truth* in the CIS by translating relevant changes from the simulated world to messages globally understood in the CIS.

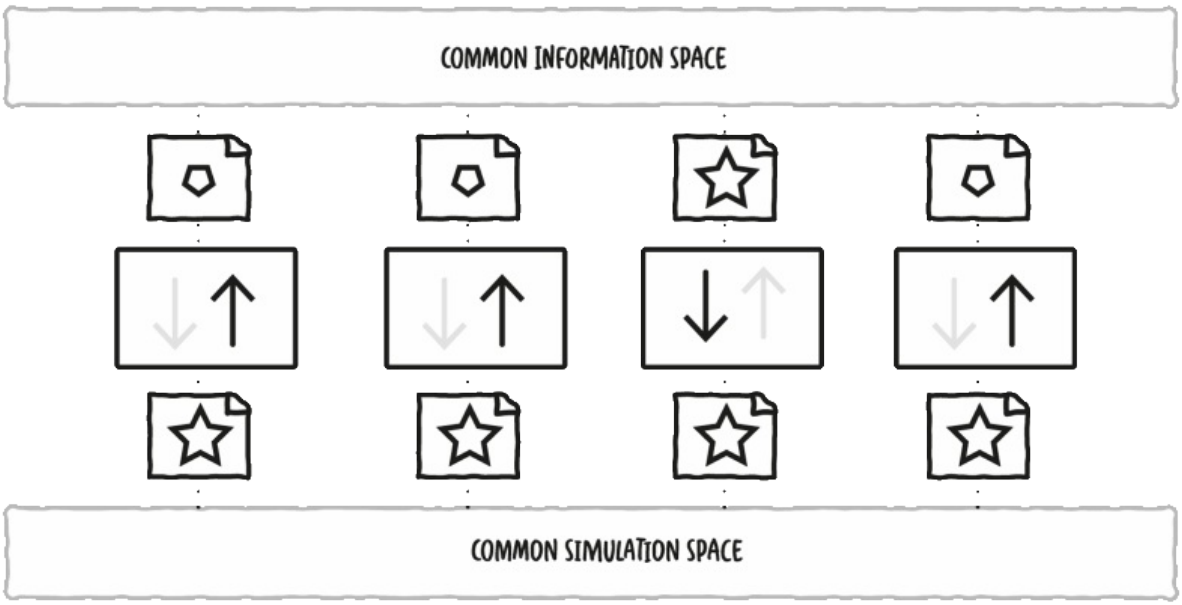


Figure 8. Conceptual diagram of gateways translating messages back and forth between CIS and CSS.

> An example of this would be the simulation of a flooding. Imagine a river that has rising water levels due to an increase of rain water. At the river bank, there are several sensors that measure the actual water level. 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 level is calculated in a flooding simulator, which connects to the CSS. The gateway listens to the calculated water levels in the CSS, uses it to compute the water levels at the exact sensor locations, and sends out the expected messages to the CIS, similar to the operational application. The decision-support tool, which is connected to the CIS, listens to these

formatted messages of the gateway as if it was connected to the actual operational application. This also provides Trial owners the option to experiment with sensor failures, more or fewer sensors, etc.

There are also Solutions that send messages which serve as commands or requests to change the simulated world. Again, a gateway would be used to bridge the CIS and CSS spaces.

> For example, a new dispatch centre Solution allows users 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 translates it to a request for changing the simulation space. The responsible simulator would receive this request via the CSS and simulate an ambulance driving through the simulated world towards the incident location.

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 is not *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.

3.2 Test-bed administration tool

The Test-bed is a collection of services, simulators and solutions running in a distributed network environment, so it is difficult to understand what is going on exactly. At the same time, the Test-bed needs to be prepared before the Trial can take place. You can compare it to a theatre play, where the stage needs to be prepared, the actors and musicians must be ready, as well as the light and sound engineers. The Test-bed administration tool, or admin tool (see Figure 9) helps you set the stage, monitors both the CIS and the CSS to better understand the status of all services, simulators and Solutions. In particular, it allows you to:

- Publish all AVRO schemas (see the sub-section on Message types) that will be used during a Trial.
- Determine whether all services are up and ready (via the adapter's heartbeat message).
- Make sure that all adapters connect to the expected topics.
- Check the distributed error log to see if any adapter is experiencing problems.
- Setup security, if required, using the `mode` dropdown in Figure 9.
- Manage multiple configurations, for different Trials, using the `configuration` dropdown in Figure 9.

If all checks are passed, it sends out an 'all-clear' message, and the Trial can start.

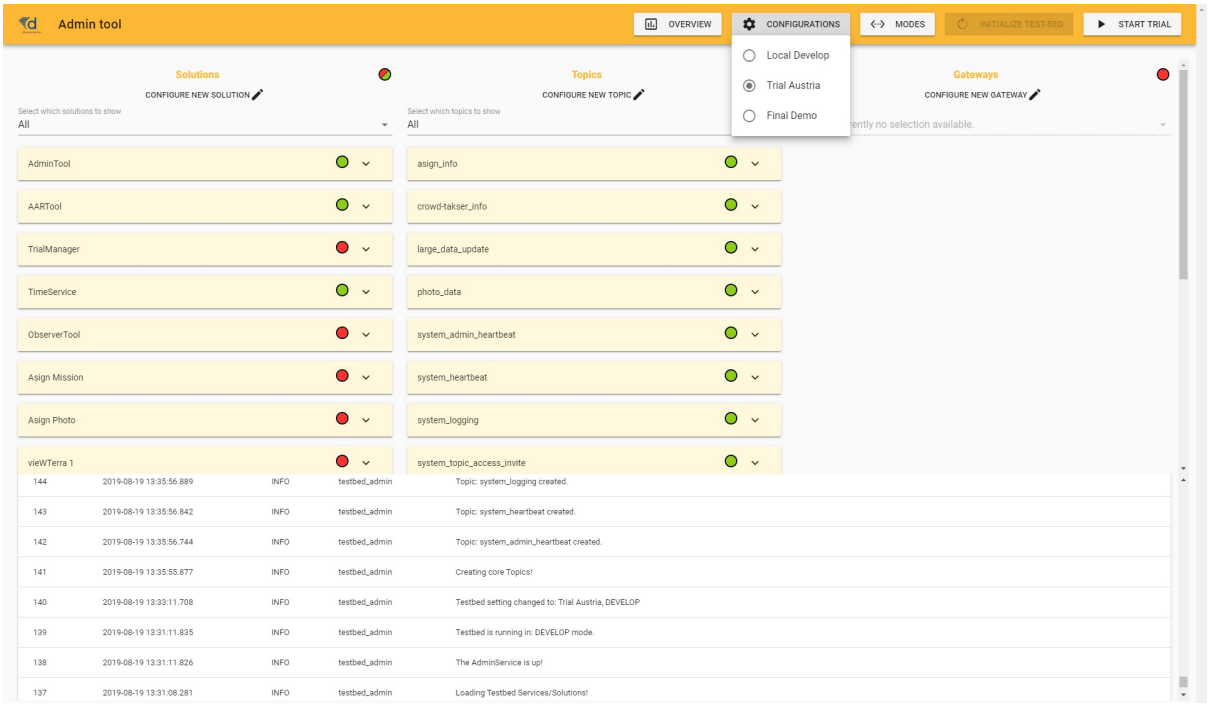


Figure 9. Selecting a different configuration in the Admin tool.

Detailed information

- Functional specification
- Website

3.3 Trialling, Exercising and Scenario Management

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

In a **Trial**, the primary objective is to test and evaluate solutions, and a similar procedure is followed. In that case, the *training objectives* are replaced by *research questions*, but the other steps remain the same. More details about the methodology to setup a proper Trial can be found in the Trial Guidance Methodology handbook, available at https://www.driver-project.eu/trial-guidance-methodology/.

Scenario support during a Trial or exercise

A Trial or exercise requires a relevant context for the participants in which they can trial new solutions or improve their competencies in handling an incident. This context is not static, but evolves in time, and may contain the incident but also non-participating organisations, and is called a *scenario*. For example, there is a storm, high waters and heavy rainfall, increasing the risk of flooding. Due to a traffic incident involving a truck, the gates of the main sluice cannot be closed anymore, and the water is threatening the inner city. Two hours later, a small dyke leakage floods an electricity station, rendering the pump to keep the polder dry useless. Etc.

In the simplest cases, such a scenario can be controlled via info cards, and at each step, a new info card message is presented to the participants. Such a message could detail an evolving situation, or inform them of a specific need, to name but a few. This quickly becomes unwieldy when dealing with many messages, or with alternative *what if* branches, and a tool would be needed to support the staff. A few tools exist to manage and support a scenario, which will be discussed in the next session. Ideally, such a tool should offer support to Trial staff for:

1. Creating and editing a new scenario, where a scenario is represented as a sequence of specific messages over time. These messages are often called *'injects'*, as they inject certain behaviour into a running Trial.
2. Providing a good overview of the scenario timeline: *what is happening when*.
3. Managing stakeholder and objectives for a Trial or exercise, and checking that these objectives are represented in the scenario.
4. Managing the requested observations: since a scenario defines what is happening when, it also 'knows' best what should be observed, and should be able to inform observers to look out for certain behaviour.
5. Including additional information into the recorded message sequence for after-action review, such as dedicated messages for starting and stopping a scenario (a session) or a CM phase.
6. A clear separation between different 'storylines', e.g. one main storyline for the evolving incident, another storyline to train organisation A, and another for organisation B. Or storylines dedicated to trialling a new solution.
7. Branching i.e. a scenario is a kind of tree where the participants decide which path to follow.
8. Conditional execution of messages or branches.
9. Support for CM messages.
10. Automation, such as automatically sending emails or tweets.
11. Running distributed, so the staff can access it remotely and edit it at the same time.
12. Controlling simulators, like a traffic simulator responding to a 'create traffic incident' message.
13. Replacing simulators when the simulation requirements are very basic and require no, or very limited, interaction, e.g. replace a flooding simulator with a pre-recorded time sequence of flood maps.
14. Managing one or more scenarios in the context of a Trial or exercise. For example, a flooding Trial may have two scenarios, and before and one after the flooding incident itself.

Existing software for managing Trials or exercises

In NATO, military exercises are often supported by the Joint Exercise Management Module, or JEMM (see Figure 10). It is a web-based tool to support live exercises as well as table-top exercises, from a few people to battalions. Essentially, it is an enhanced spreadsheet, where each line represents an action, and the time moves down vertically, making it difficult to separate multiple storylines or see the whole picture. It also 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, the Command Staff Trainer of the Dutch Army employs a dedicated simulator to simulate the outcomes of a battle. This simulator is controlled manually, however, and JEMM is only used to inform a human operator when he or she has to take a particular action.

Since JEMM is only available to military NATO members, in which case it is free-of-charge, to use it in a Trial would require military involvement.

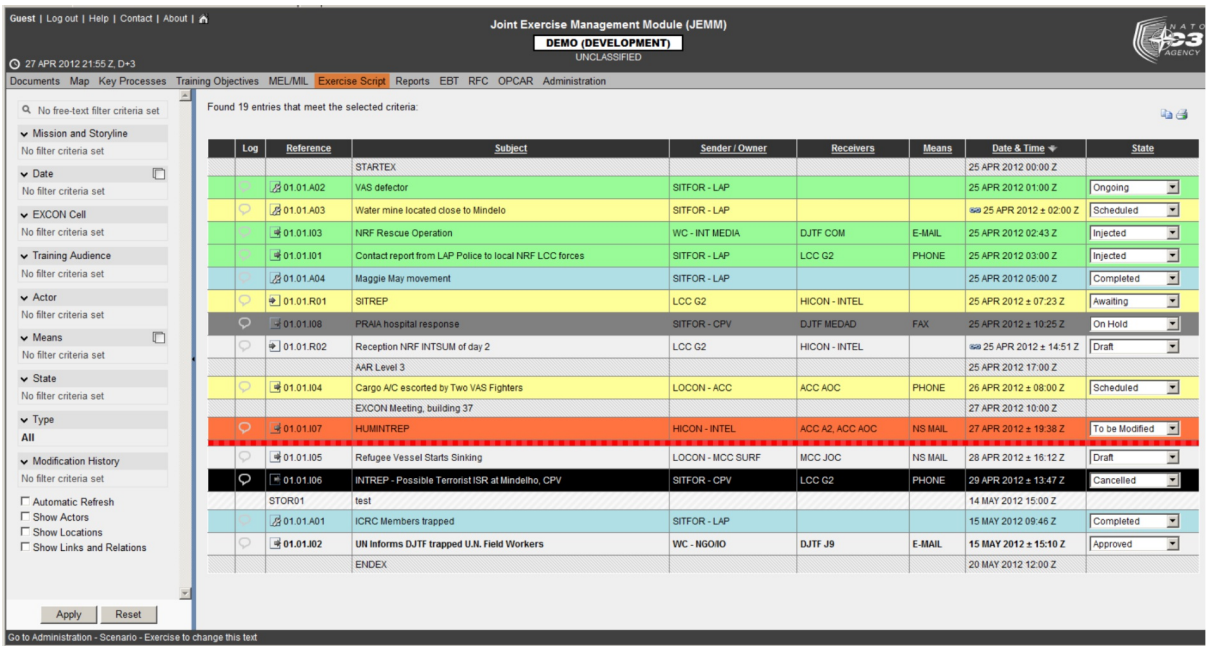


Figure 10. JEMM exercise script example.

Alternative commercial solutions exist too, such as Exonaut (see Figure 11). They also focus on a military audience, and follow a similar approach.

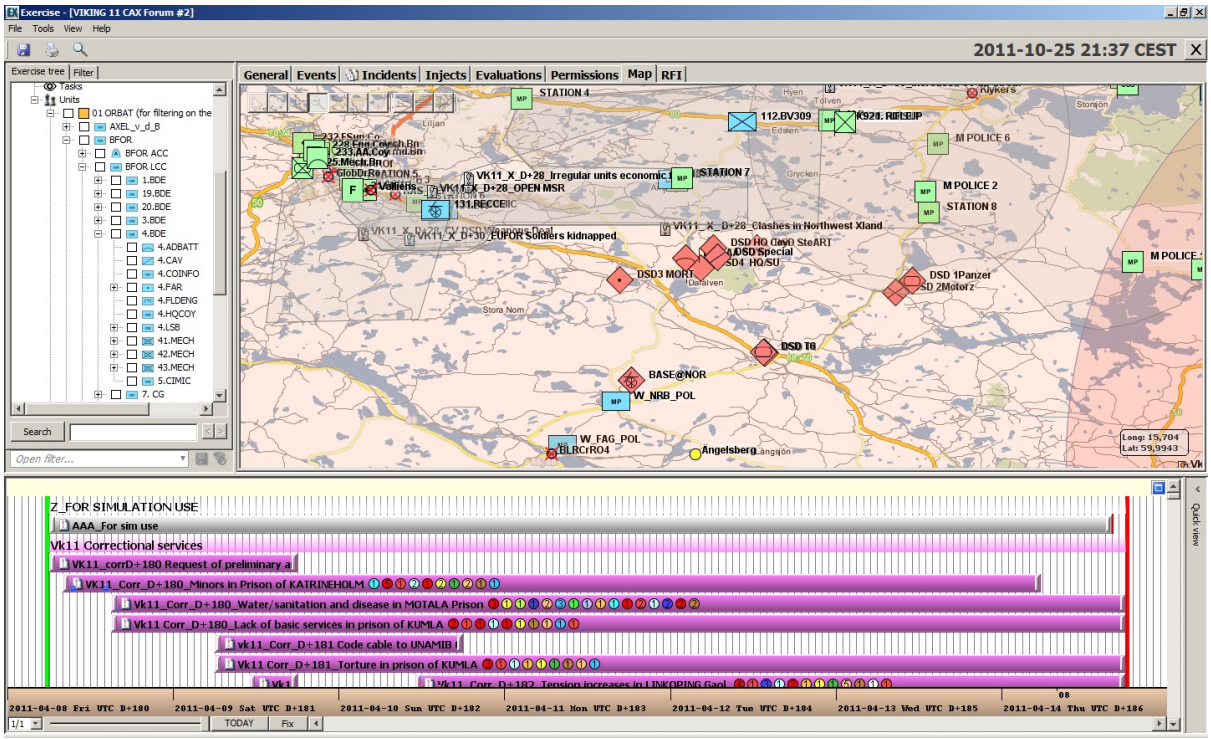


Figure 11. Exonaut timeline example.

Scenario management using the Trial-Management-Tool (TMT)

To properly support the complex Trials in DRIVER+, tooling is required for managing the scenario. However, it is not possible to use JEMM or Exonaut directly, as:

- JEMM is only available to NATO members, and can only be used in an exercise when military personnel requests it. This will often not be the case in CM-centred Trials which are the focus of DRIVER+.
- JEMM and Exonaut are aimed at the military community, and the fit with the Crisis Management domain is not optimal. For example, there is not support for typical CM messages like CAP, and contains a lot of military jargon.
- JEMM and Exonaut are closed source, so a strong integration with the Test-bed is not possible, as the applications cannot be modified.

Within the Test-bed, the open source Trial-Management-Tool is an integral part, and it allows the Trial staff to fulfil all requirements as stated above. For any Trial or exercise, one starts by defining the *training objectives*, *What does the training audience need to learn?*. Next, one or more appropriate *scenarios* are formulated by CM experts in which these training objectives can be tested and exercised, providing a realistic context (see Figure 12). The scenario is further broken down into *storylines*. A storyline describes a developing situation that trigger a trialled solution or will set conditions and provide the training audience an opportunity to achieve specific training objectives. It often targets a subset of the solutions or training audiences, e.g. only the COP tool or 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. Additionally, the scenario may include extra messages for managing the Trial, such as the time that a scenario was started or stopped (a session), instructing the observers to pay attention to certain behaviour, or annotation the begin and end of a CM phase.

During the Trial execution, those messages influence the participant's context. For example, the TMT can send a message to a traffic simulator to create a traffic incident at a certain location, or it could send a Common Alerting Protocol (CAP) message to a Command & Control application. Additionally, the TMT can send messages to role-players, so they can make a call or play a non-participating command centre. The Trial staff can also send messages earlier or later, or resend them, offering a great level of control over the Trial.

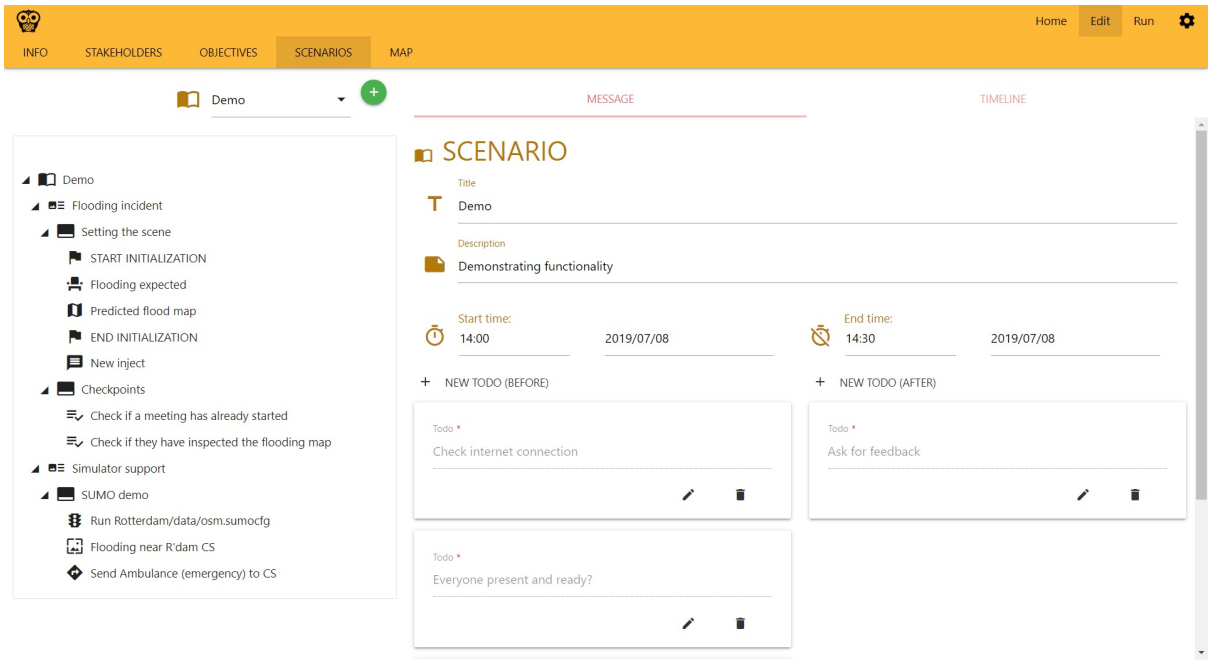


Figure 12. Edit a scenario, including specifying start and end time and creating checklists.

Creating a scenario in the TMT can be compared to creating a new project (see Figure 13). However, instead of managing a project by creating subprojects, work packages and tasks, a Trial scenario (=> project) consists of storylines (=> subprojects), acts (=> work packages) and injects (=> tasks, like a simple message). And whereas in a project, you assign resources, in the TMT you assign simulators, role players and observers.

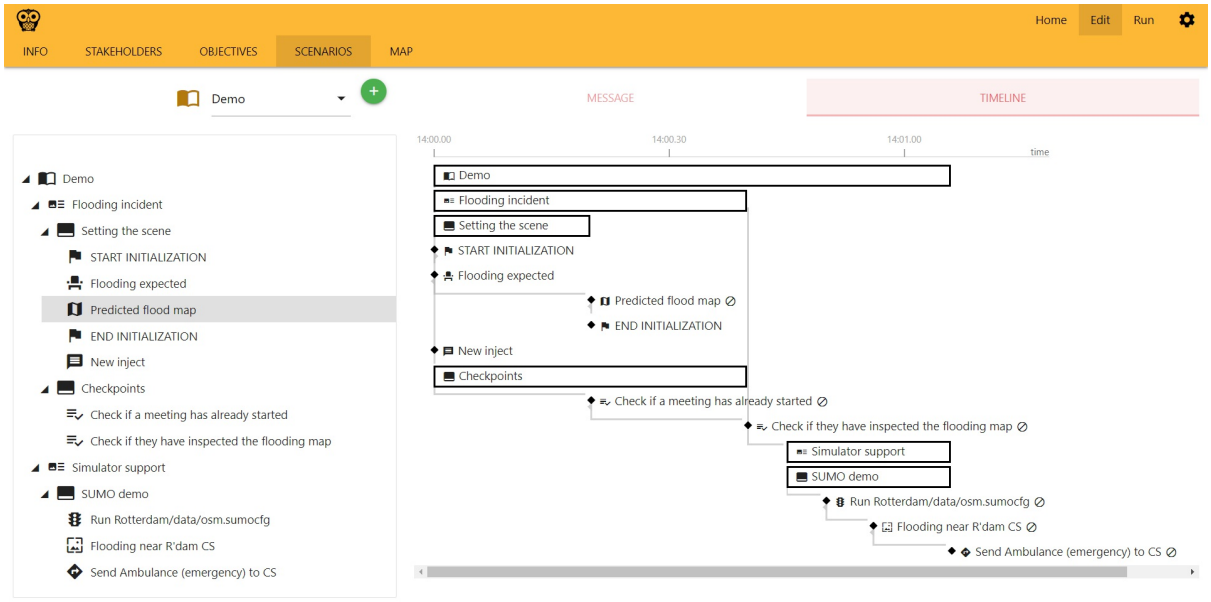


Figure 13. A scenario timeline is comparable to a project manager's Gantt chart.

A scenario is created while preparing the Trial and executed during the Trial (see Figure 14). And like a project manager controlling the sequence of the tasks during the lifetime of a project, the Trial Director is also able to control the sequence of injects/messages during the lifetime of a scenario.

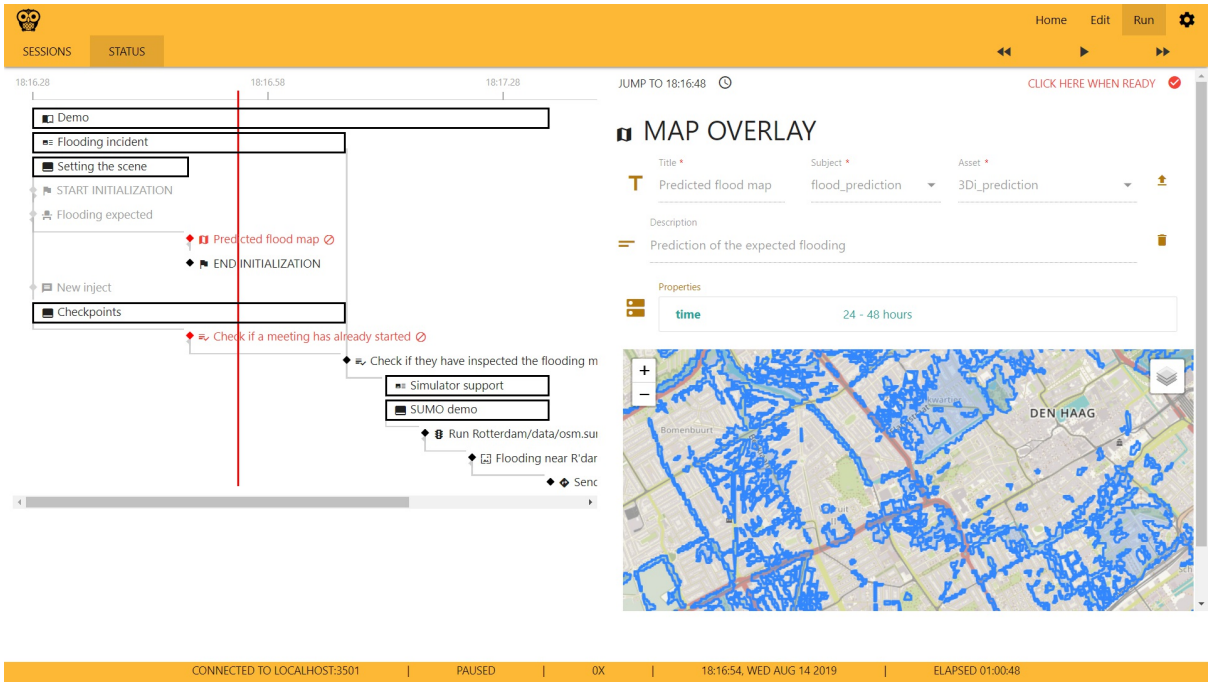


Figure 14. A running scenario can be paused, and certain messages require manual confirmation.

Detailed information

- Functional specification

- Website

3.4 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.

Observer Support Tool (OST)

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.

The Observer Support Tool, therefore, records all observations from the observers digitally, so they can be analysed during and after the Trial. To collect feedback, the OST also provides the ability for participants and Trial staff to file questionnaires, directly after (a part/episode of) the Trial is executed.

The OST consists of a web application for the observers that is typically run from a tablet. The same application can also be accessed on a browser or a normal laptop or desktop computer, for instance for participants to fill in the questionnaires and for the Evaluation coordinator to prepare the Trial specific observation templates (i.e. checklists) and questionnaires. Furthermore, a server is running to manage all the checklists and questionnaires and record all the answers. This server is connected to the Trial-Management-Tool, such that the correct checklists/questionnaires are available at the applicable moments during execution of the Trial. All collected observation and questionnaire data is thereafter shared with the After-Action-Review tool, such that it is centrally stored for evaluation.

Although the observer tool (see Figure 15) 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 record the observations so they can be used during the analysis and evaluation.
- The Trial-Management-Tool 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 D923.21 - Trial guidance methodology and guidance tool specifications (version 1).

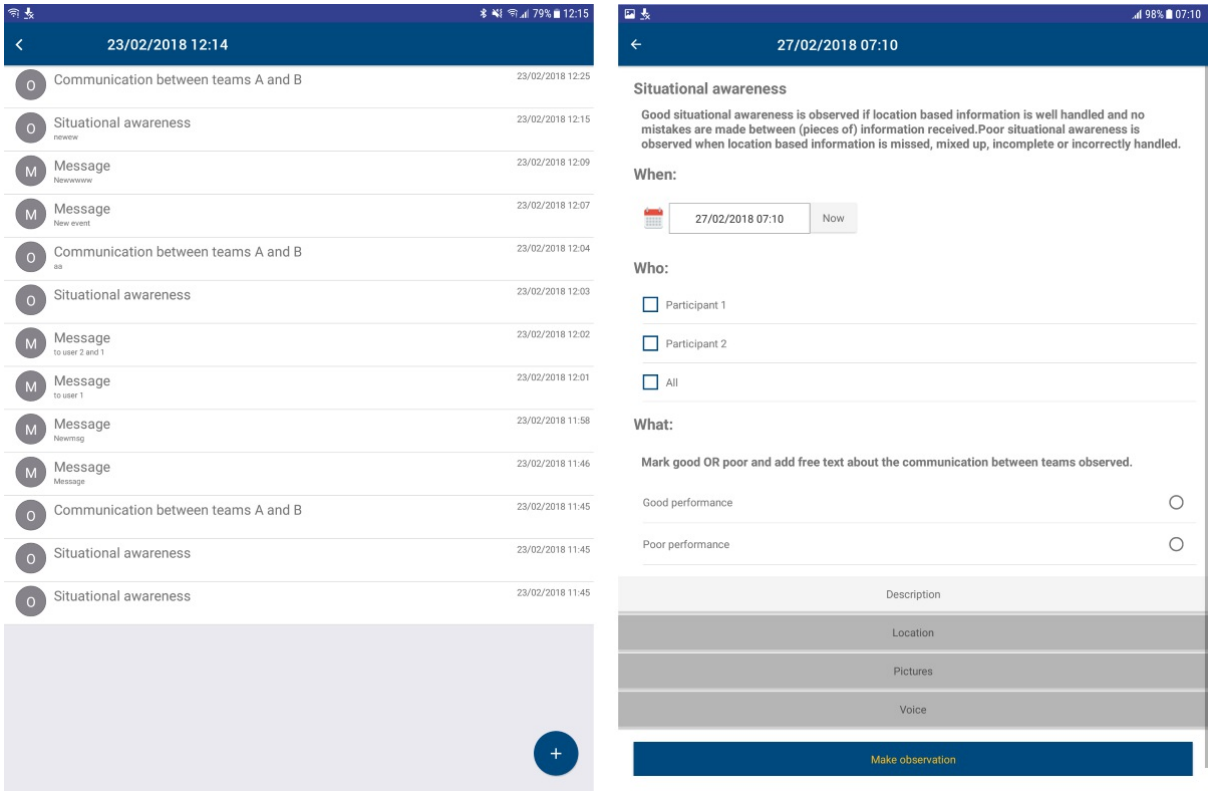


Figure 15. Observer Support Tool: Left, an overview of available observation templates. Right, one of the observation templates is selected.

Detailed information

- Functional specification
- Website
- Documentation

After-Action-Review tool (AAR)

A successful Trial partially depends on the ability to analyse the Trial in depth afterwards. The After-Action-Review (AAR) tool, therefore, collects all data during a Trial: messages exchanged during the Trial between solutions, legacy systems and simulators, observation reports, and screenshots of the solutions during the Trial. 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. Using a timeline displaying the collected messages, the whole Trial can be analysed visually, and one can quickly jump to a specific point in time in order to inspect a specific message, observation or see the active screenshot (see Figure 16).

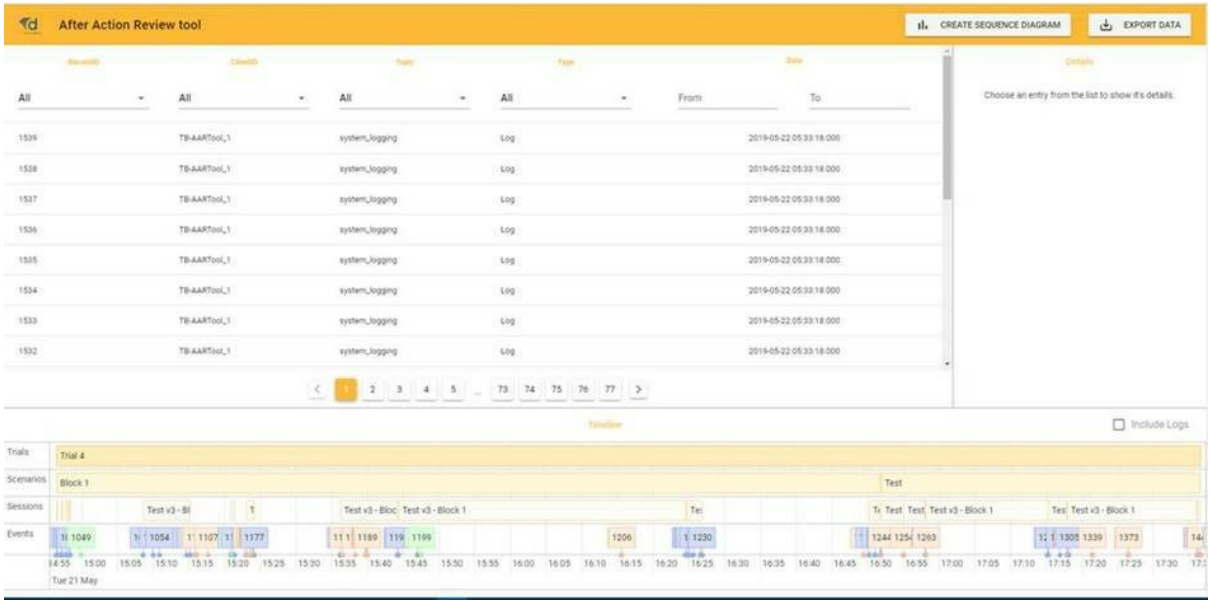


Figure 16. In the AAR, messages can be filter, inspected and made visual on the timeline.

Apart from being used for a post-analysis, it is also used during a Trial preparation and execution to monitor the amount and kind of data exchange, in order to check whether all data exchanges are correctly functioning, to check whether the correct data is exchanged at the correct moment during scenario execution and to check whether observations are being stored.

The detailed logging of all formats, sources and destinations, all marked with time-stamps, allows the technical staff to sort, filter and inspect the messages. The output of the message logging can be viewed on a list, on a timeline or as a sequence diagram. This enables several options for a visual analysis about which components have exchanged which data with each other. For example, one can use a sequence diagram (see Figure 17) to show the message exchange between certain solutions.

Figure 17. AAR sequence diagram, showing the interaction between filtered systems.

Detailed information

- Functional specification
- Website

3.5 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. For example, during a CM exercise, it took the participants quite some time to figure out that the water was actually threatening their own location, and they had not 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 and GUI (see Figure 18): 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 Trial-Management-Tool, as discussed above.

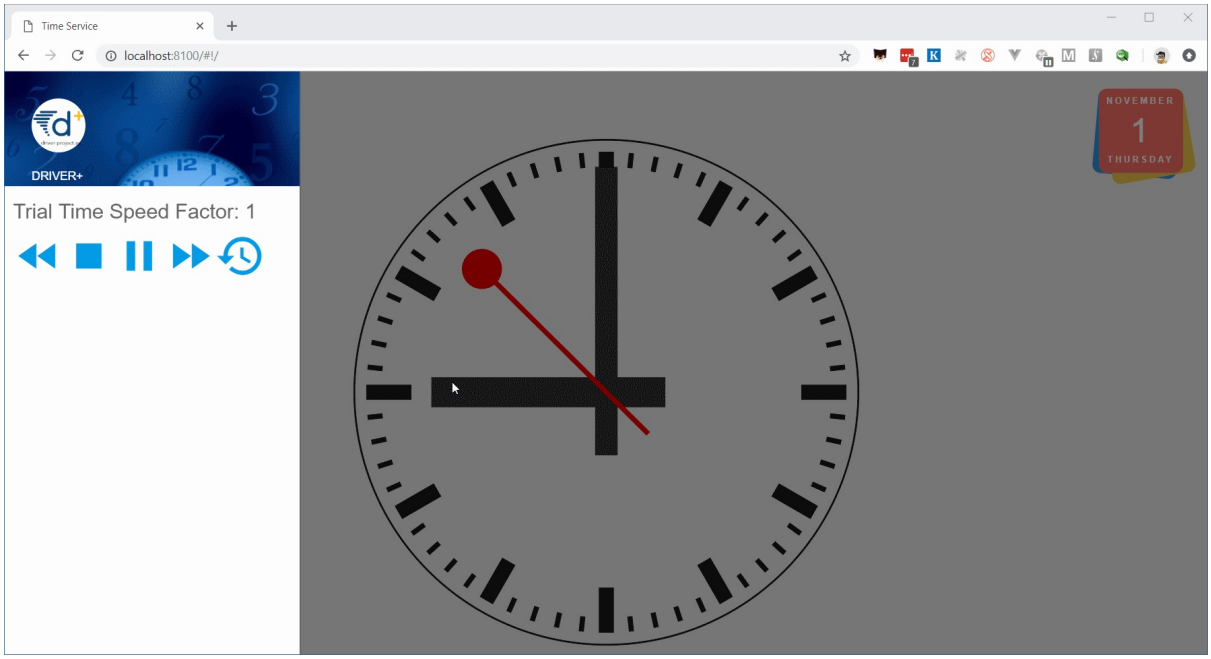


Figure 18. In case no TMT is running, the Test-bed time-service GUI can also be used to start/stop/pause a Trial, as well as set the simulation speed.

It does not, however, provide these simulators as an integral part of the Test-bed. They are, and shall always remain, external. And even though some simulators will be connected during the project, they are 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 SE-STAR to the Test-bed, e.g. providing a realistic simulation of people in need during a crisis.
- TNO connects their critical infrastructure and chain effects simulator to the Test-bed, providing insights into the cascading effects of failing infrastructure.

In a recent paper at the SUMO User Conference in 2019, Co-simulation of vehicles and crowds for rescue trials, the connection between XVR OnScene (3D view), SUMO and SE-STAR has been demonstrated and explained in depth.

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.

3.6 Conclusion and Roadmap

The actual Test-bed reference implementation v3 already provides all the required functionality to conduct Trials for testing the usefulness of new solutions for CM organisations. But CM organisations more often train than trial, and future versions should enhance the training capability too, improving support for table top and field exercises. Preferably, this should be done in close cooperation with the DRIVER+ Centre's of Excellence, typically CM umbrella organisations.

Besides being useful for CM organisations, there is also a growing need in the Defence domain for urban battlespaces. Whereas battles of the past were mostly conducted in the open, modern battles are typically conducted in an urban environment, and kinetic simulators and training environments must be enhanced with urban environments and their typical characteristics: critical infrastructures, social media, cyber and sensor networks. The Test-bed is, therefore, also useful in the Defence domain.

Finally, also the police has an increasing need for such tooling: terrorist attacks, manhunts or large scale public events are often not isolated to a single location and may generate multiple incidents, requiring regional and national cooperation. Having an environment to train such large scale incidents should be beneficial for them too. Ideally, required functionality would be specified together with the Police Academy.

It would even be interesting to see if the Test-bed, and especially the TMT, can be used to recreate the timeline in crime-fighting. Take the example of an assassination, abduction or finding a missing child: create a main storyline based on the known facts. For each hypothesis, add a new storyline that fills in the gaps and analyse it. Does the alibi of X hold, or may he have met Y somewhere in between. Get a better impression of the relevant time period when watching public CCTV camera's or reviewing social media photos.

These combined needs may lead to the following future functionality.

Test-bed

The Test-bed could be enhanced with additional services, some being actively developed, such as:

- An email gateway service, to forward Test-bed messages to email clients and receive emails. In case emails are based on fixed templates, these templates can be interpreted by a computer and trigger events or sent to other solutions.
- A geo-fencing service, triggering events when a participant, resource or simulated entity cross a pre-set geographic location.
- Gateway to HLA (especially the REAPER FOM) or DIS, to improve interoperability.
- Map services for sharing maps.

- A communication service so voice and chat messages can be recorded.
- Adapters in Rust, C++ and Go.
- Text-to-speech services, so phone calls can be generated and replace role-players.
- Speech-to-text services, so phone calls can trigger events, e.g. a phone call including a menu, requesting a dyke break at a certain location in order to spare the city.

The Admin tool already provides most required functionality, and required enhancements are documenten on GitHub. They are mostly minor things, like improved editing of the configuration, add sorting or filtering of info logs.

The TMT's enhancements are documented on GitHub. In particular:

- Adding new message types should be simpler.
- Allowing for life edits such that the staff can inject new messages on the fly.
- Allow to start a scenario 'in the middle', i.e. skipping part of the scenario. For example, to jump to an interesting scene or during debugging or integrating.
- Listening to external messages, like a trigger from the geo-fencing service, informing the TMT that something relevant has occurred, or a voice command via phone or chat to start a storyline.

For the OST, the main focus is on better user-friendliness, making it easier to create new observations, and improved analysis capability. And finally, for the AAR, support capturing screenshots or video from running solutions.

4. The Test-bed for developers and system administrators

This chapter discusses the Test-bed from a technical perspective, and is aimed at skilled IT persons, such as system administrators and developers. It is presumed that you are already familiar with the previous chapters.

- A *system administrator*, in the current context, is responsible for installing the Test-bed on their local network or in the cloud, 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.

Watch the following animation (also available at https://player.vimeo.com/video/299681354) to get a better understanding of what is going on.

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, which also contains many example configurations as applied during the DRIVER+ Trials. Alternatively, use the GUI to build your own Test-bed environment in Docker.

System administrators are also responsible for setting up the local network and firewall settings, 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 a developer can proceed to integrate a solution or simulator with the Test-bed reference implementation.

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. When running locally, a good place to start is to checkout the local docker-compose.yml with the following command: `docker-compose up -d`. Next, access the Admin tool and initialize the Test-bed, so all schemas are made available in the schema registry.
2. Developer selects one of the existing adapters, and runs its examples: each adapter has a producer, that sends messages, and a consumer, that listens to them. Adapters are available in multiple languages, each adapter is published separately as a library

or package, and contains an example project explaining how to use it: Java, C#, JavaScript/TypeScript/Node.js and separate example project, Python and REST. The Admin Tool shows whether the connection is established successfully. In the Kafka Topics UI, you can inspect the messages that you've sent.

3. Developer uses the selected adapter, integrates it into his application, and connects to the Test-bed. Integrating is simple: when publishing information, create the message (object) and use the adapter's publish method. For receiving messages, just add a handler when subscribing to a topic. This handler is invoked when a new message is received.
4. 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, the developer needs 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. Please consult the adapter's documentation.
    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. See the paragraph on the Gateway service below.
5. To receive messages, the developer uses the example project's producer or starts up the Kafka replay-service to send messages one-by-one or replay a logged sequence of messages. For more detailed information, see the Replay service section below. To log the messages in a topic, you can use the Kafka-topics-logger or the Kafka topics UI, which is a default part of the Test-bed, 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.
6. In case the integrated application also sends messages, Developer can use the Kafka topics UI in Figure 19 to verify that they were created and published to the Test-bed successfully.
7. When your message uses time, the developer needs to query the adapter to get the local Trial time. The Test-bed's time-service manages the Trial time, and you can read more about it below.

Kafka Topics UI

The Test-bed includes Landoop's Kafka topics UI service (see Figure 19) to inspect all the message topics that exist (default location http://localhost:3600). For each topic, you can see inspect the messages that were sent. This is useful to verify your success in sending a message to the Test-bed, and to check the exact message contents.

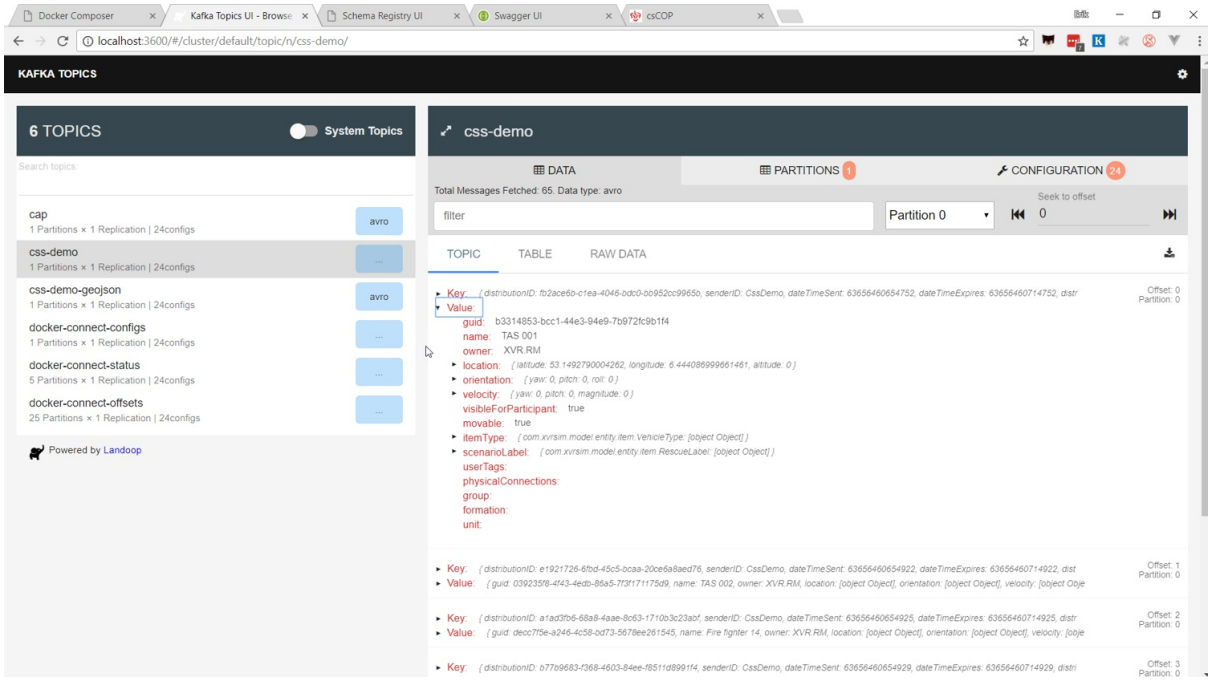


Figure 19. Screenshot of Landoop's Kafka topics UI, which is part of our test-bed.

AVRO Schema Registry

The Test-bed also includes Landoop's AVRO schema registry service (see Figure 20) 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 one of the existing adapters.

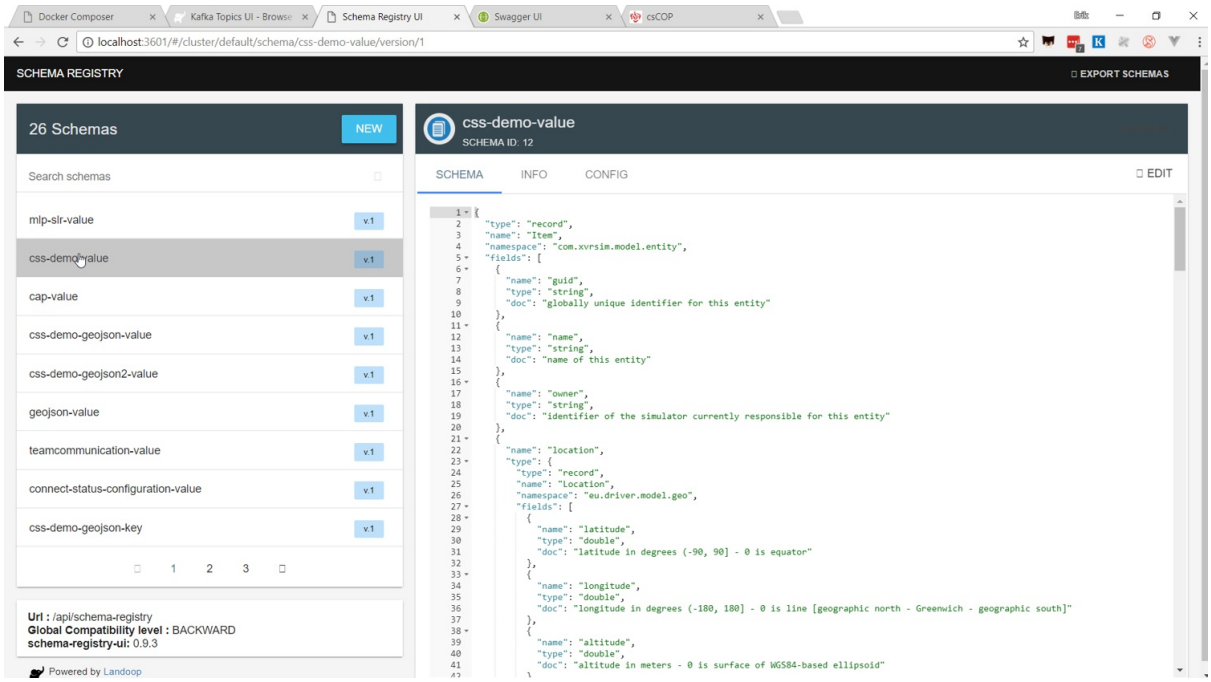


Figure 20. Screenshot of Landoop's AVRO schema registry, which is part of our test-bed.

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 currently supported, it may be necessary to interact with the Test-bed via the REST service. In that case, the REST service would be installed and run locally, so the solution that connects to it is still clearly visible in the admin tool as a separate client.

Replay service

Connecting to the Test-bed is needed to share information that you produce, or to 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 21) to present you with messages to consume. This can be either a single message or a sequence of messages.

> 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, log all the messages to file using our Kafka-topics-logger or via the Kafka Topics UI. This log file is subsequently sent to the COP solution developer, who can replay it using the replay service, so the developer can work as if the simulator was present.

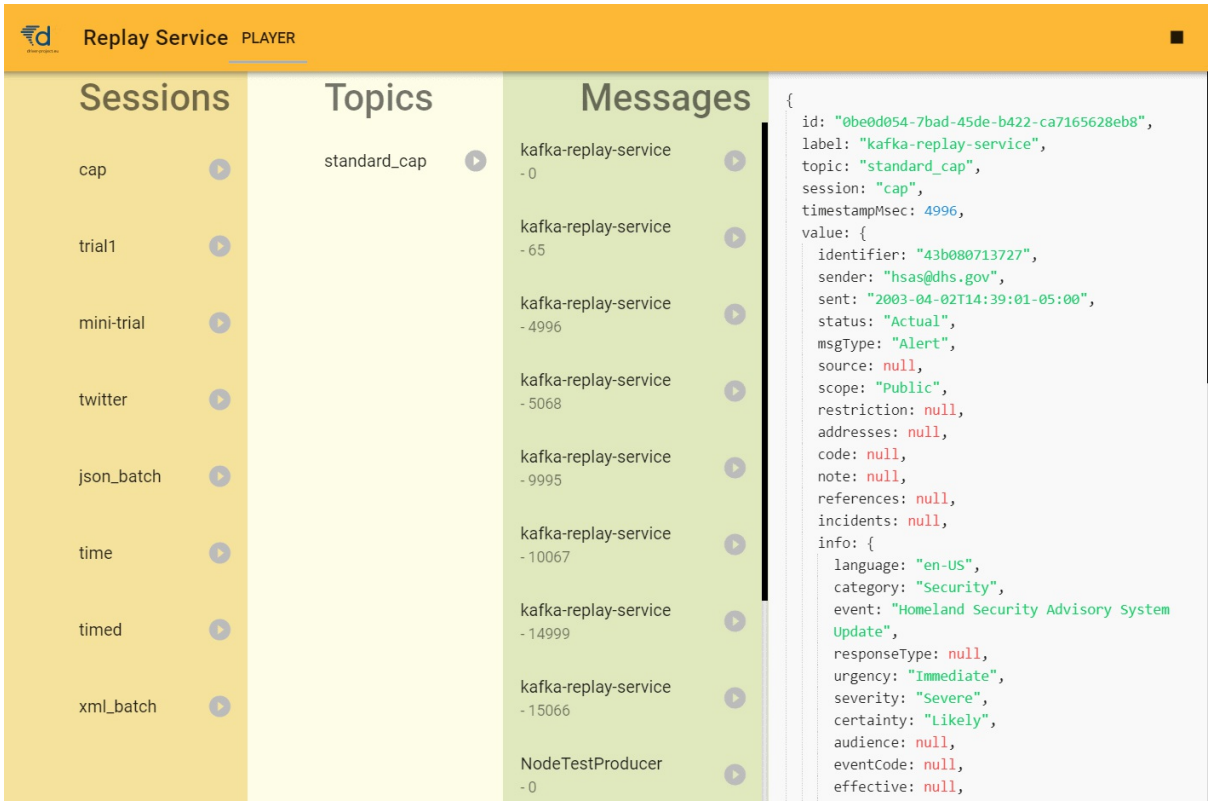


Figure 21. Screenshot displaying the Replay service's GUI interface.

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. System administrator 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 by the Admin tool with the schema registry.
2. System administrator inspects the Admin Tool and verifies that all required solutions and simulators are available and running online without errors. Via the Admin Tool, solutions and simulators may be invited to subscribe to certain topics.
3. System administrator starts up the Trial-Management-Tool (TMT), 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. System administrator selects a Trial scenario and creates a new session in the TMT, 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. System administrator resets the Trial scenario and stops the session (so the AAR can finalize the session, and clearly mark all messages that belong to the current session).

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.

> For example, the Twitter-gateway translates messages from the Test-bed to Twitter, ready to be consumed via Twitter, or the XVR-GeoJSON-gateway converts XVR simulator messages to the GeoJSON standard.

4.5 Large file service (LFS)

Kafka and AVRO are ideally suited for smaller-sized messages, i.e. typically not exceeding 1Mb compressed. However, a typical flooding file or a drone video file may well exceed 1Gb of data. Instead of sending these large messages, adapters can upload a file to the Large File Service (LFS) and send out a large-data message sharing its location. Alternatively, the URL may be embedded in a standard document, like a CAP message.

The LFS currently supports two types of messages:

- Public messages: they can be browsed in a folder like structure.
- Private messages: although not secret, they are shared via an obfuscated URL, which is very hard to guess, similar to services like wetransfer.com, so you get security through obscurity.

In most adapters, there are facilities to support the LFS: for example, the adapter can automatically send a notification message (large-data-message) after successfully uploading a file to the LFS.

4.6 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, which makes the information available to legacy systems too.

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.

4.7 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.

Since a Trial can encompass several time zones, the trial time is always communicated in UTC (Universal Time Coordinates). Depending on your application, you may need to convert this to local time.

Within the Test-bed, therefore, the trial time (a.k.a. Trial time or fictive time) is controlled via the time service (see Figure 22) using two types of messages: one for controlling the time, and one for informing adapters about the current trial 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 trial 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 trial 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 (in UTC).
  - **Initialized:** the scenario is ready to be started. All adapters will receive the scenario start time, and can use this to initialize their service.
  - **Running (started or paused):** the trial time is moving forward, either at normal speed (1x) or slower/faster than normally. In case the scenario is paused, the current trial 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.

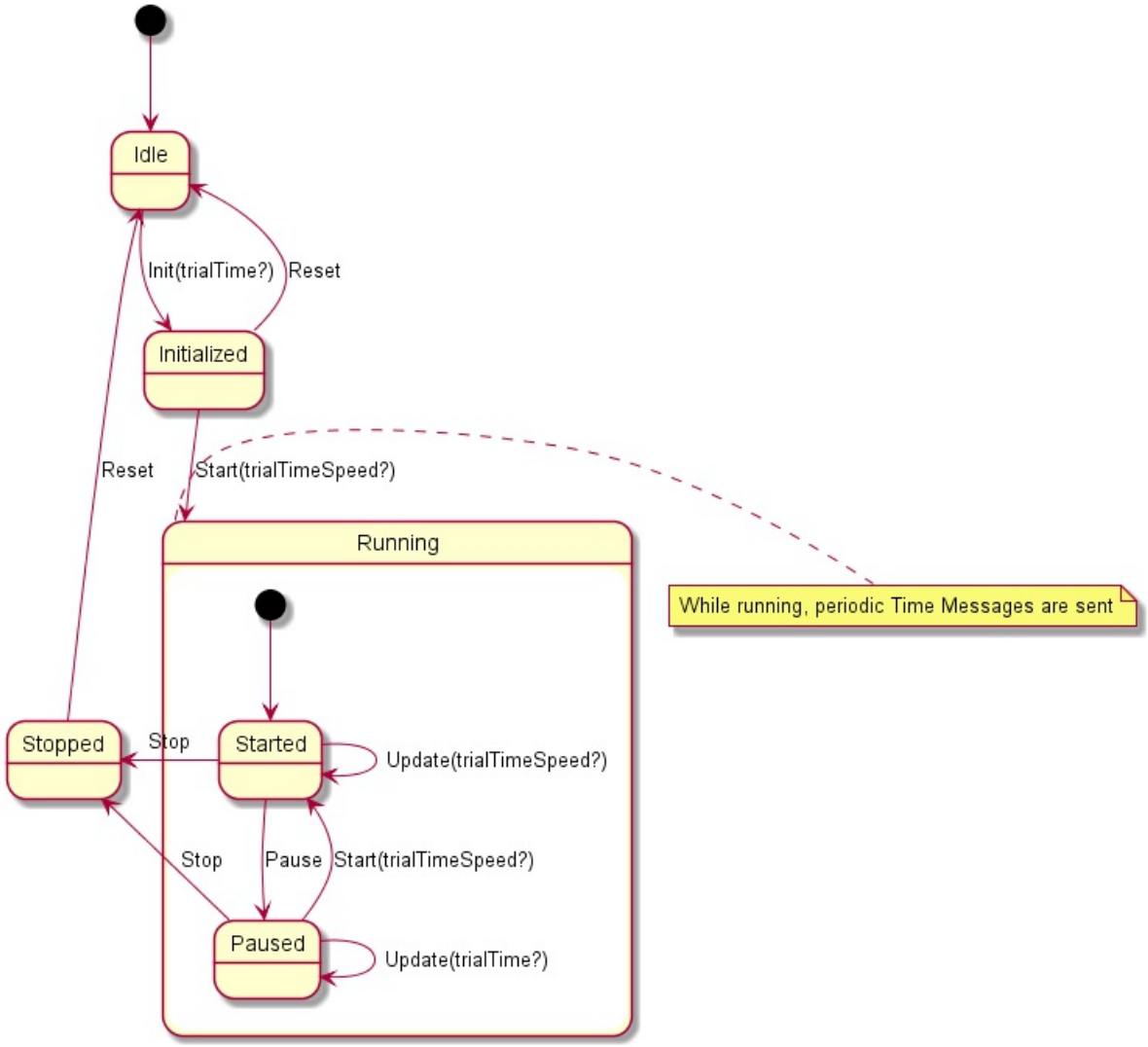


Figure 22. State diagram of the time service.

> **NOTE:** Even when you do not interface with the time messages, you still must use the time interface when you want to publish a message with a timestamp inside. This timestamp should use the current trial time. Similarly, in case you display the 'actual' time in your user interface, please also use the current trial time.

System time versus trial time

In Trials executed in the past, the operating system (OS) time was also adjusted to match the trial time. The advantage was that if you would check the time in your status bar, it would display the current trial 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 is why the current Test-bed does not require you to synchronize your system time to the trial 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, C++ or C#, and there is less support for the 'newer' programming languages, like JavaScript, Python, etc.
- HLA requires a run-time infrastructure, which is a kind of test-bed: there are two commercial providers, but both their products are expensive. Although there is one open source version, it is feature incomplete and not well maintained. Furthermore, even though different products should be interoperable, they are not, and they cannot be mixed easily.

That is 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 and solve issues themselves.
- New schemas for your message, or create them when needed.
- 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/DIS 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 if suboptimal, in practice, this is not really noticeable.

4.7 Security

Security is an integral part of the Test-bed: although in most crisis management organisations, practitioners consider that safety prevails anyhow, often to such an extent that security never comes into the picture or merely as a measure of last resort. However, a lack of security controls in crisis management solutions from the ground up may have a disastrous impact on crisis management decisions, operations and, in the end, on people's lives; therefore, DRIVER's test-bed has *security by design*. Our primary focus is on the security of information exchanged in CIS and CSS topics, because this is where critical or sensitive information flows, leading to three information security objectives:

- Integrity of information exchanged in CIS and CSS topics: messages should not be tampered without notice.
- Authenticity of information exchanged in CISS and CSS topics: the origin of the messages published to certain topics should be authenticated, and, by extension, only legit/trusted users may publish on certain topics.
- Confidentiality of information exchanged in CISS and CSS topics, such as critical infrastructure information or personal

information (beware the GDPR): only authorized Solutions (adapters) may have access to the messages published to certain topics.

To achieve these objectives, the Test-bed includes an access control framework in which both system administrators and developers have a role to play.

Authentication

The CIS/CSS Kafka broker enforces SSL/TLS transport security with mutual authentication. This means that adapters are required to authenticate with SSL client certificates. Developers shall get such SSL client certificates from the Admin Tool, with the approval of a system administrator. The subject name of the certificate must uniquely identify the adapter within the organization. More specifically, the subject name must include an Organization name (O) identifying the Organization that owns the Solution or Simulator, and a Common Name (CN) identifying the Solution or Simulator instance within that organization. The Admin Tool uses a Certificate Authority (CA), provided with the testbed in the backend, to issue those certificates.

Developers then configure their adapters with those certificates as client certificates (key store), the aforementioned CA's certificate in their truststore (trusted CAs) in order to authenticate the broker (using a certificate issued by that same CA as well), and recommended TLS parameters according to current best practices: TLS v1.0 or later (as of writing), strong cipher suites, etc. Developers must take good care of protecting the confidentiality of the private key associated to their client certificate according to best practices and the risk level. If the key is compromised, authentication is useless. Developers must report any compromised key to system administrators so that the certificate be revoked.

By default, the CIS/CSS broker trusts only that backend CA for client certificates. If developers wish to use client certificates issued by another CA, system administrators may grant their wish by modifying the broker's SSL configuration. This change consists to add the other CA's certificate to the broker's SSL truststore (JKS) and restart the broker service.

Once all adapters are properly configured by developers for SSL client authentication, they are able to connect to the secured CIS/CSS broker and at least access public topics such as *system_* topics.

Authorization

For stronger security, whenever trial owners want to protect specific CIS/CSS topics, system administrators shall enable **topic authorization**. By default, this test-bed security feature is disabled, i.e. all authenticated users (according to previous section) have access to all topics. System administrators enable topic authorization by using an extra Docker Compose file that extends the default one with authorization enforcement components.

Once enabled, access to any topic is denied by default, except for certain public topics such as *system_* topics. This means that for other topics, access must be granted explicitly by system administrators. More specifically, for each sensitive topic X, the system administrator shall configure the topic access policy via the Admin Tool. A topic access policy is set on a specific topic X and consists of a list of access rules. Each access rule consists of:

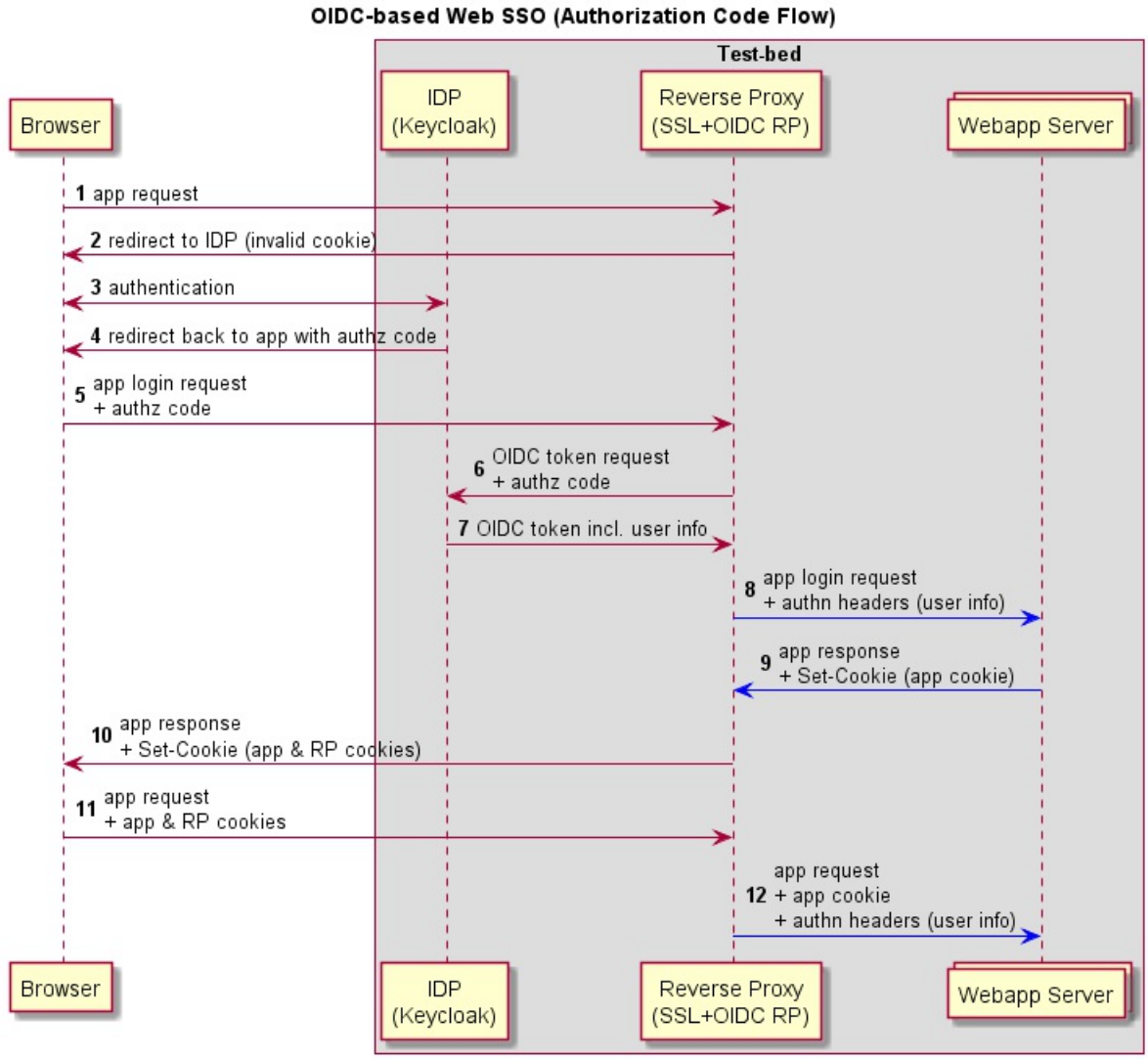
- Authorized subject name, as in the adapter's SSL certificate (a.k.a. Kafka client ID in this case), OR Kafka consumer group ID;
- A list of *permissions*; each *permission* is a couple *(ACTION, bool)*, where *ACTION* is the action considered on the topic (in Kafka API model), e.g. *READ* (subscribe a topic) or *WRITE* (publish on a topic), and the Boolean *bool* is true if and only if the action is permitted (positive rule), else denied.

If a consumer group ID is used as the first item (instead of subject name), the system administrator must also declare the corresponding group memberships in the Admin Tool, i.e. who is authorized to join this group.

In order to improve performances, system administrators can enable the authorization decision cache (disabled by default) in Kafka's configuration via Docker Compose environment variables. This allows Kafka (the Kafka authorizer in particular) to cache decisions to avoid requesting the remote PDP every time. The downside is that changes to the topic access policies that occur in the remote PDP during the cache interval may be ignored.

Single-Sign-On (SSO)

While securing the CIS/CSS Kafka topics is important, the Test-bed also exposes several web clients, such as the Admin Tool, the TMT, AAR, and OST. Especially the AAR needs to be protected, otherwise all exchanged messages can be inspected openly, and securing the topics is useless. If the Admin Tool, TMT or OST are left unprotected, outsiders can, even unwittingly, disrupt a Trial, for example when trying out some functionality or stopping a scenario session. Therefore, the web clients need to be protected too. One approach is to add authentication and authorization to each of these services separately, but this would imply that Trial staff and developers need to manage multiple users accounts. Instead, the Test-bed makes use of a so-called Single-Sign-On setup based on the open source Keycloak service, so Trial staff only need one username and password to access all Test-bed services, as explained in the high-level diagram below.



Red arrow: HTTPS
Blue arrow: HTTP (no SSL)
OIDC: OpenID Connect
IDP: Identity Provider, aka OpenID Connect Provider
RP: (OpenID Connect) Relying Party, aka OAuth Client
App: Webapp Server
authn: authentication
authz: authorization

4.8 Online Test-beds

As mentioned in section 4.3, it is possible to make use of an online test-bed that is hosted in the cloud. This serves as an alternative to hosting a test-bed yourself and is useful for hosting the Test-bed as a service and integration testing: whenever two or more parties wish to test their integration via the test-bed, they can request a cloud-hosted test-bed that is available within a few minutes. All parties can then remotely connect to this test-bed, without requiring any prior setup and configuration. Additionally,

during several Trials, some solutions required the use of an externally hosted backend service that needed to communicate with the Test-bed. In that case, an online Test-bed is also useful in order for the backend service to access the Test-bed so the Trial organisation does not have to open up their firewall for allowing access.

The current Test-bed cloud consists of four servers configured as a Docker Swarm which allows various Test-bed compositions to be hosted in parallel. So not all running instances need to be the same, as the Trial requirements may vary. Docker swarm allows any Test-bed docker composition to run on the Docker Swarm servers. Each Test-bed can make use of one of the ten available host-names for temporary online test-beds: https://tb1.driver-testbed.eu to https://tb10.driver-testbed.eu.

4.9 Reverse Proxy for Test-beds

Optionally the Driver Test-bed can be configured to work with a Reverse proxy like Nginx, Traefik or Redbird. This has several advantages:

- The Test-bed Services (i.e. Topic UI, Schema UI, etc) are not separately exposed on their own port, but via a relative URL on a shared proxy that can be reached via a configurable hostname. So instead of exposing the Topics UI at `http://hostname:3601` and the Schema UI at `http://hostname:3602`, they can be exposed at `http://hostname/topics-ui/` and `http://hostname/schema-ui/` respectively. They will then both share the same HTTP endpoint at `http://hostname`.
- The shared HTTP endpoint can be secured with an SSL certificate, like the free certificates created by lets-encrypt, which all above-mentioned reverse proxies support out-of-the-box. Because all web services in the test-bed use the same reverse proxy, securing this reverse proxy entry point will secure the HTTPS traffic for all services behind it without requiring any configuration on the test-bed services.
- Reverse proxies monitor and log requests, and therefore allow for tracing and monitoring health and load on services in the test-bed.

> As an example, Traefik is built to integrate seamlessly with Docker Swarm. For setting up the Test-bed with Traefik these instructions are followed. Traefik runs in a separate container, on a shared Docker network with the Test-bed containers to allow routing of traffic to the Test-bed services. It listens to events from Docker stating that a container is started. Metadata provided on start-up of the container (labels, see here) specifies how the service should be exposed by Traefik. For example, Traefik can expose the service at a specified hostname (domain or subdomain) and at a specific relative path. Moreover, the URL may be manipulated (for example strip the relative path) to allow compatibility with the services. Whenever a new service is hosted on a specified hostname, Traefik automatically ensures that a valid lets-encrypt certificate is present or requested.

There are two beta Test-bed compositions available for usage with Traefik on the test-bed repository `traefik` branch:

- A composition for a local test-bed. This allows running the test-bed on your local machine. The external hostname, Kafka broker port, and Schema Registry port can be specified in the `.env` file.
- A composition to run in the cloud using Docker swarm. This is built for running on the TNO-hosted cloud servers. The external hostname, Kafka broker port, and Schema Registry port must be specified using ENV variables `TESTBED_HOST`, `BROKER_PORT`, and `SCHEMA_REGISTRY_PORT` respectively.

Abbreviations

Abbreviation	Definition
AAR	After Action Review tool, which supports reviewing a Trial when it is finished
AVRO	Open data serialization system supported by the Apache Organisation. avro.apache.org
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 Annex 1
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 en.wikipedia.org/wiki/Distributed_Interactive_Simulation
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, docs.oasis-open.org/emergency
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 en.wikipedia.org/wiki/High-level_architecture
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 CM solutions, their application, strength and weaknesses, and user reviews.
Python	Programming Language
REST	Representational State Transfer (common interface allowing you to read and write data from a service)
RTI	Run-time infrastructure, a kind of HLA-based Test-bed
SA	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.
TMT	Scenario Management Tool, allows you to create new scenarios

TGM	Trial Guidance Methodology, as defined in deliverable D922.21, "Trial guidance methodology and guidance tool specifications (version 1)".
TGT	Trial Guidance Tool, an online application that supports a practitioner in setting up a Trial
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

In order to have a common understanding within the DRIVER+ project and beyond and to ensure the use of a common language in all project deliverables and communications, a terminology is developed by making reference to main sources, such as ISO standards and UNISDR. This terminology is presented online as part of the Portfolio of Solutions and it will be continuously reviewed and updated[1]. The terminology is applied throughout the documents produced by DRIVER+. Each deliverable includes an annex as provided hereunder, which holds an extract from the comprehensive terminology containing the relevant DRIVER+ terms for this respective document.

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.	ISO22300:2018 (en)
Evaluation	Process of estimating the effectiveness, efficiency, utility and relevance of a service or facility.	ISO 5127:2017(en) Information and documentation — Foundation and vocabulary, 3.1.3.02.
Exercise	Process to train for, assess, practise and improve performance in an organization. Note 1 to entry: Exercises can be used for validating policies, plans, procedures, training , equipment, and inter-organizational agreements; clarifying and training personnel in roles and responsibilities; improving interorganizational coordination and communications; identifying gaps in resources; improving individual performance and identifying opportunities for improvement; and a controlled opportunity to practise improvisation. Note 2 to entry: See also test.	ISO22300:2018(en) 11
Experiment	Purposive investigation of a system through selective adjustment of controllable conditions and allocation of resources. DRIVER+ Note 1: Please consider also the notes in the original definition. DRIVER+ Note 2: In the context of DRIVER+ Trials are executed - and not experiments.	ISO/TR 13195:2015(en) Selected illustrations of response surface method — Central composite design
Trial	An event for systematically assessing solutions for current and emerging needs in such a way that practitioners can do this following a pragmatic and systematic approach.	Initial DRIVER+ definition.
Trial Guidance Methodology (TGM)	A structured approach from designing a Trial to evaluating the outcomes and identifying lessons learned	Initial DRIVER+ definition
Trial Guidance Tool	A software tool that guides Trial design, execution and evaluation in a step-by-step way (according to the Trial Guidance Methodology) including as much of the necessary information as possible in form of data or references to the Portfolio of Solutions	Initial DRIVER+ definition
Legacy systems	(Crisis management) system currently in operational use.	Initial DRIVER definition
Observer	Observer participant who witnesses the exercise while remaining separate from exercise activities. Note 1 to entry: Observers may be part of the evaluation process.	ISO 22300:2018(en) Societal security — Terminology.

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.	Initial DRIVER+ definition.

[1]. The Portfolio of Solutions and the terminology of the DRIVER+ project are accessible on the DRIVER+ public website https://www.driver-project.eu. Further information can be received by contacting coordination@projectdriver.eu. ↩

Annex 2 Change history

This annex documents the changes with respect to previous versions. The returning reader is therefore urged to only review the recent changes.

Version 2

The changes with respect to the first version are described in the following.

Executive Summary

- Added a reference to the change history.
- Added link to deliverable D922.21 which is now publicly available.
- Added link to Test-bed animation.
- The Python adapter has become available.
- Added links to the eBook versions (PDF, epub and mobi formats).

Introduction

- Added a reference to the change history in section 1.3.
- Clarified the scope and aims of the document, based on the EU reviewers comments.
- Removed a duplicated picture.

A Test-bed for Crisis Management Practitioners

- Before discussing the Test-bed, first discuss it from the perspective of the Trial owner (chapter 2 and 3 were swapped with respect to the previous version, thereby addressing the reviewers comments).
- Added an description of the Test-bed composer, in order to distinguish it better from the PoS of TGT.
- Embedded Test-bed overview animation (only in the online version).
- Scenario manager use case is expanded.
- Rewrote the use cases, to enhance the role of Monica.
- Removed a duplicated picture.

Test-bed description

- Added a brief description in section 2.1 on how to trial non-technical solutions, i.e. processes.
- Rewrote subsection Adapters and added two new features: security and support for large files.
- Messages: added link to our public AVRO-schemas repository.
- CIS and CSS: wording has slightly changed.
- Gateways and Validation Services: improved the clarity.
- In section 2.2 Test-bed administration tool: improved the wording.
- Added a new image for the Gateway.
- In section 2.3 2.3 Trialling, Exercising and Scenario Management:
    - Added link to TGM and functional specification document.
    - Added link to Trial-Management-Tool and AAR website.
    - Added screenshot of time service.
    - Mentioned TNO's critical infrastructure simulator.

Test-bed for Trial developers & sysops

- New section on Security
- New section on the Large File service
- Embedded Test-bed developer animation (only in the online version).
- Added link to the report "Solution testing procedure".
- Added link to the Python adapter which has become available.
- Added information about two gateways on GitHub (Twitter gateway and XVR-GeoJSON gateway).
- Updated information.
- Updated screenshot of the Replay service.
- Added section on online Test-beds and reverse proxy.

Test-bed design

- Moved to Annex 3.
- Moved section on large data to Chapter 4.
- Truncated section on the Adapters, and referred to Chapter 3.

Version 3

The changes with respect to the second version are described in the following.

Overall changes

- Renamed Scenario Manager to Trial-Management-Tool: the reason for this change was that a Trial may contain several scenarios.

Executive summary

- Updated to present version 3.
- Added a paragraph on additional software that we delivered, and which is not part of the original DoW.

Introduction

- Added a reference to the change history in section 1.3.

A Test-bed for Crisis Management Professionals

- Included a new diagram with the high-level overview of all Trial components (Figure 4).
- Included a remark that the Test-bed is also useful for supporting table top or field exercises, even if this is out-of-scope.
- In Use Case 2.1, added a manual Test-bed creation mode. In practice, Test-beds are mostly created by technical people, and they prefer to work directly with the text-based Docker images files.
- In Alternative 1, Cloud scenario, the actually implemented cloud scenario is added. It turned out that the effort required for the ideal situation was too high, compared to the low effort needed for the current implementation.
- In Alternative 2, Local scenario, the current way-of-working is added.
- Slightly updated Use Case 2.2 to reflect the latest state.

Test-bed description

- Added a conclusion / roadmap section at the end.
- Added a short description on how the Test-bed can be used for trialling processes and procedures. The previous version was a bit negative about its use, but that vision has changed due to more recent experiences.
- Added an overview of the type of messages that can be exchanged.
- Added direct link to TGM handbook.

- Rewritten the part about the TMT.
- AAR text updated, added images.
- Added two papers (ITEC2019 and SUMO conference)
- Minor textual edits.
- Updated most images.

The Test-bed for developers and system administrators

- Improved intro, added link to node.js example project.
- Added more detail to use case 4.2.
- Added section on Single-Sign-On as part of section 4.7.
- Rewritten the paragraph 'Dealing with standards' to better explain why we are using AVRO.
- Added a new section: 'Scenario support during a Trial or exercise'.
- Updated the section on JEMM. Added example of the Command Staff Trainer.
- Removed reference to the message injector.
- Renamed Message topics UI to the Kafka topics UI.
- Renamed Schema registry to AVRO schema registry.
- Time service: added a remark that we are always communicating UTC time.
- Time: improved consistency, scenario time is always referred to as trial time.
- HLA: There is nowadays the option to use C#.
- Minor textual edits throughout the chapter.

Annex 3: Test-bed design

- Minor textual edits throughout the chapter.

Annex 3. 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.

Annex 3.1 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.

Annex 3.2 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.

Annex 3.3 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 is 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.

Annex 3.4 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.

To read more about the implementation, see Chapter 2 Adapters.

Annex 3.5 Security architecture

The testbed Security Architecture consists of the following elements:

- Security Services.
- CIS/CSS broker security features.
- CIS/CSS adapter security features.

For more details, read on the next sections.

Security Services

There are two security services running in the test-bed:

- Certificate Management Service.

- Authorization Service.

Certificate Management Service

The Certificate Management Service provides:

- A Certificate Authority (CA) that issues SSL/TLS client (resp. server) certificates to the CIS/CSS adapters (resp. broker).
- A REST API - backed by the aforementioned CA - for managing the lifecycle of X.509 (standard) certificates used for SSL/TLS authentication in client-server communications on the testbed. This is the API used by the Admin Tool on behalf of system administrators, to issue SSL client certificates certificates to developers for their adapters. The API is described on the test-bed's GitHub repository.
- An endpoint for verifying certificate revocation status (CRL or OCSP standard). This is the endpoint used by the CIS/CSS broker and adapters to validate the status of the certificate of any remote peer they are establishing a SSL session with.

The Certificate Management Service is based on the open source project EJBCA (Community Edition).

Authorization Service

The Authorization Service provides two endpoints:

- PAP (Policy Administration Point).
- PDP (Policy Decision Point).

The Authorization Service is based on the open source XACML 3.0 PDP implementation project AuthzForce (Thales).

Policy Administration Point

The PAP (Policy Administration Point in ABAC (Attribute-Based Access Control) standard model) endpoint exposes a REST API for managing CIS/CSS topic access policies. System administrators interact via the Admin Tool with this API for defining and managing the access policies for CIS/CSS topics. The API is described on the Authorization Service's GitHub repository.

Policy Decision Point

The PDP (Policy Decision Point in ABAC (Attribute-Based Access Control) standard model) exposes a REST API for requesting an authorization decision (Permit/Deny) according to the policies defined via the PAP, and an input authorization request. The CIS/CSS broker (via its PEP, aka Kafka *authorizer* in this case) interacts with this API for deciding whether to reject (if PDP returns Deny) or let a message go (if PDP returns Permit) through the broker. We say that the CIS/CSS broker enforces the PDP's decision.

The PDP interface complies with the standards:

- REST Profile of XACML 3.0 for the transport protocol (HTTP) and entry point;
- JSON Profile of XACML 3.0 for the API request-response payload formats (XACML Request/Response).

CIS/CSS broker security (Kafka)

The broker security consists of the following:

- Kafka SSL/TLS layer.
- DRIVER+'s custom Kafka *authorizer* (XACML-driven).

Kafka TLS layer

The TLS term is used instead of SSL *because we mandate the use of TLS 1.0 or later in DRIVER+.*

The Kafka broker exposes a TLS server interface that enforces the confidentiality, integrity and mutual authentication of communications with CIS/CSS adapters. It authenticates to Kafka consumers with a TLS server certificate issued by the Certificate Management Service's CA mentioned earlier. The Kafka broker is configure to require TLS client certificate authentication with client

certificates issued by that same CA. The TLS setup follows the official Kafka documentation's section: Encryption and Authentication with SSL.

Custom Kafka Authorizer (XACML PEP)

The Kafka broker's configuration is modified to use a custom Authorizer implementation that can interact with the Authorization Service's PDP endpoint mentioned earlier. This Authorizer plays the role of PEP (Policy Enforcement Point) in the ABAC (Attribute-Based Access Control) standard model, and more specifically in the XACML standard architecture. Therefore, this custom Authorizer supports the required XACML standards (REST and JSON profiles) and rejects (resp. accepts) messages if the PDP's returned decision is Deny (resp. Permit).

In order to improve performances, this Authorizer may cache decisions to avoid requesting the remote PDP every time. The downside is that changes to the topic access policies that occur in the remote PDP during the cache interval may be ignored. This decision cache is enabled and configured via Docker Compose environment variables.

The Authorizer's source code is open and available with technical documentation on a dedicated GitHub repository.

CIS/CSS adapter security

CIS/CSS adapters are Kafka clients. Therefore, they are required to use TLS (1.0 or later) for communicating with the CIS/CSS Kafka brokers, and TLS client certificates (X.509) provided by the aforementioned Certificate Management Service. In order to properly authenticate the CIS/CSS broker, adapters must use a TLS truststore (set of trusted CAs) that contains the certificate of the Certificate Management Service's backend CA, or preferably of the Root CA (that the backend CA is a subordinate of).

The Kafka documentation gives an example of Kafka client SSL configuration.

Annex 4. Docker example for starting a Test-bed

A functional Test-bed requires many services, and installing them physically on a machine would require a substantial amount of time, and is error prone. Additionally, not every Trial or exercise requires the same setup, and therefore the Test-bed uses the virtualisation services provided by the open Docker platform.

To execute a successful Trial, you just need to combine the services that you need in a simple text file, often called `docker-compose.yml`, after which you can start all Test-bed services with a single command.

```
# For starting all services:
> docker-compose up -d
# Or to stop it and clean up everything, run:
> docker-compose down
# It removes everything so you can start with a clean slate,
# except for the recorded messages, which use a so-called 'volume.
```

A minimal example is given below, but actually used versions during the Trials can be found at the Test-bed website at https://github.io/DRIVER-EU/test-bed. In the example, besides Apache Kafka for sharing information, and Apache Zookeeper for keeping track of client sessions, you see the schema registry for registering all used schemas, a time service, and the Trial Management Tool. The large file service is commented out (not needed at the moment). Typical Test-beds are more extensive, and contain additional tools such as the admin tool, AAR, OST, schema registry UI, and several more.

```
---
version: '3'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:5.0.0
    hostname: zookeeper
    ports:
      - "3500:3500"
    environment:
      ZOOKEEPER_CLIENT_PORT: 3500
      ZOOKEEPER_TICK_TIME: 2000

  broker:
    image: confluentinc/cp-kafka:5.0.0
    hostname: broker
    depends_on:
      - zookeeper
    ports:
      - "3501:3501"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:3500'
      KAFKA_ADVERTISED_LISTENERS: 'EXTERNAL://localhost:3501,PLAINTEXT://broker:9092'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: 'EXTERNAL:PLAINTEXT,PLAINTEXT:PLAINTEXT'
      KAFKA_LISTENERS: 'EXTERNAL://0.0.0.0:3501,PLAINTEXT://0.0.0.0:9092'
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_DEFAULT_REPLICATION_FACTOR: 1
      KAFKA_MESSAGE_MAX_BYTES: 100000000
      KAFKA_REPLICA_FETCH_MAX_BYTES: 100000000

  schema_registry:
    image: confluentinc/cp-schema-registry:5.0.0
    hostname: schema_registry
    depends_on:
      - zookeeper
      - broker
    ports:
      - "3502:3502"
    environment:
```

```
      SCHEMA_REGISTRY_HOST_NAME: schema_registry
      SCHEMA_REGISTRY_LISTENERS: 'http://0.0.0.0:3502'
      SCHEMA_REGISTRY_KAFKASTORE_CONNECTION_URL: 'zookeeper:3500'
      SCHEMA_REGISTRY_KAFKASTORE_BOOTSTRAP_SERVERS: 'PLAINTEXT://broker:9092'

  time_service:
    image: drivereu/test-bed-time-service:latest
    depends_on:
      - broker
      - schema_registry
    ports:
      - "8100:8100"
    environment:
      KAFKA_BROKER_URL: broker:9092
      SCHEMA_REGISTRY_URL: http://schema_registry:3502
      AUTO_REGISTER_SCHEMAS: 'true'

  # large_file_service:
  #   image: drivereu/large-file-service:latest
  #   ports:
  #     - '9090:9090'
  #   environment:
  #     HOST: localhost
  #     PORT: 9090

  trial_management_tool:
    image: drivereu/trial-management-tool:latest
    depends_on:
      - broker
      - schema_registry
    ports:
      - '3210:3210'
    environment:
      CLIENT_ID: TB-TrialMgmt
      KAFKA_HOST: broker:9092
      SCHEMA_REGISTRY: http://schema_registry:3502
      TRIAL_MANAGER_SERVER_PORT: 3210
      PRODUCE: system_request_change_of_trial_stage,system_tm_phase_message,system_tm_role_player,system_tm_session_mgm
t
      SSL: 'false'
      SSL_PFX: certs/TB-TrialMgmt.p12
      SSL_PASSPHRASE: changeit
      SSL_CA: certs/test-ca.pem
    volumes:
      - trial-data:/app/trials

volumes:
  trial-data:
```

List of Figures