Driving Innovation in Crisis Management for ***E***uropean ***R***esilience

D22.12 - DRIVER test bed: Architecture, Integration and Orchestration

Grant agreement number:	607798	Due date of deliverable: 31/07/2015
Start date of the project:	2014-05-01	Actual submission date: 31/07/2015
Duration:	54 months	

Lead Beneficiary: TNO

Contributing beneficiaries: ATOS, FOI, TCS, TNO, JRC, ARMINES, E-Semble

Keywords:

Dissemination level:	
PU	☒
PP	☐
RE	☐
CO	☐

Release History

Version	Date	Description	Release by
0.1	9 Oct 2014	Very First Draft.	TNO
0.2	21 Oct 2014	Upgrade to new DRIVER template and updates from telco 16/10/2014.	TNO

0.3	18 Nov 2014	Include Annex B: Survey on scenarios as a place holder to collect test bed use cases and scenarios to identify required services. Writing distributed to partners in telco on 17/11/2014.	TNO
0.4	28 Nov 2014	Processed task level peer-review comments and several editorials	All
1.0	28 Jan 2015	Processed review comments from external reviewers. Released as D22.11.	TNO
1.1	10 Jun 2015	D22.12 draft version for peer review within Task 22.1.	TNO
1.2	5 July 2015	D22.12 draft version with peer review comments processed.	TNO
1.3	7 July 2015	D22.12 version for DRIVER review	TNO
2.0	29 July 2015	Processed review comments from external reviewers. Released as D22.12.	TNO

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
CAP	Common Alerting Protocol
CIS	Common Information Space
CM	Crisis Management
COP	Common Operational Picture
CR	Crisis Responders
EDXL	Emergency Data Exchange Language
FOM	Federation Object Model
HLA	High Level Architecture
HLA-RTI	HLA Run Time Infrastructure
LVC	Live, Virtual, Constructive
NTP	Network Time Protocol
PFIF	People Finder Interchange Format
SDEM	Simulation Data Exchange Model
SOMA	Service-Oriented Modelling and Architecture
SOS	Sensor Observation Service
TSO	Tactical Situation Object
UML	Unified Modelling Language
VPN	Virtual Private Network
WAN	Wide Area Network

Executive Summary

The purpose of this deliverable is to describe the architecture of the DRIVER Test Bed to support DRIVER Crises Management and Crises Responders experimentation. The architecture description is developed in four iterations identified by the deliverable number D22.1n (where n is 1..4). This deliverable concerns the second iteration.

This deliverable describes:

- the objectives of the Test Bed and the approach for developing the architecture description across the four iterations;
- the architecture viewpoints that are used to describe the architecture of the Test Bed, and the distinction between reference architecture (for the Test Bed) and solution architecture (as an instantiation of the reference architecture for a certain experiment);
- the Test Bed architecture using the defined architecture viewpoints; and
- the rationale for selecting the HLA as reference architecture for connecting Test Bed tools.

The Test Bed architecture is described using the following architecture viewpoints:

- Services viewpoint: services that the Test Bed generally should provide.
- Systems viewpoint: Architecture Building Blocks required to implement the desired services.
- Standards viewpoint: standards that may influence an architecture building block.

The Test Bed adapts the so called service bus coupling approach for connecting simulation tools. This coupling approach uses the High Level Architecture (HLA) as simulation architecture. The HLA is a general reference architecture for distributed simulation and defines a service bus for connecting simulation tools. The service bus is called the HLA Run Time Infrastructure and provides a standard application programming interface that is used by Test Bed tools to coordinate their activities, exchange data, and progress simulation time.

Architecture building blocks are used to describe the Test Bed architecture and to provide specific focus on the logical aspects of the architecture. An architecture building block represents a basic element of re-usable functionality, providing support for one or more capabilities that can be realized by one or more components or products. An architecture building block will be instantiated as required to form a specific solution for a specific DRIVER experiment. A realization or instance of an architecture building block is called a solution building block, such as a certain software package.

This deliverable provides initial and preliminary information on the Test Bed architecture. Future iterations will take into account information from other DRIVER deliverables as they become available, such test bed objectives, methods, models, use cases and scenarios, and experiences from the DRIVER experiments.

1 Introduction

1.1 Purpose

This deliverable describes the recommended architecture of the DRIVER Test Bed. The architecture description is developed in four iterations identified by the deliverable number D22.1n (where n is 1..4) and this deliverable concerns the second iteration, with initial and preliminary information on test bed services.

The purpose of this deliverable and subsequent deliverables of follow-on iterations is to describe the architecture of the DRIVER Test Bed in order to provide:

- sufficient information to acquire or develop each DRIVER Test Bed element;
- sufficient information to integrate and test the DRIVER Test Bed elements;
- information on the DRIVER Test Bed internal and external interfaces.

This deliverable focuses on the identification of DRIVER Test Bed services and Test Bed Architecture Building Blocks.

1.2 Scope

The scope of the architecture description is the DRIVER Test Bed elements, the services that the DRIVER Test Bed elements offer, and the interfaces and interface mechanisms they provide to test bed operators, simulations tools, and operational systems.

1.3 Test bed objectives

The Test Bed objectives are described in the DRIVER deliverables D21.2n (where n is 1..3), titled "State of the Art and Objectives for the DRIVER Test-bed". The top level Test Bed objective is to support experimentation, that is, to support the exploration and demonstration of new crisis management solutions (CM Solutions) within the crisis management system of systems, for simplicity called the "System of Interest".

The System of Interest consists of real people, real organizations, operational tools, procedures, etc., located in some real world Environment as illustrated in Figure 1.1.

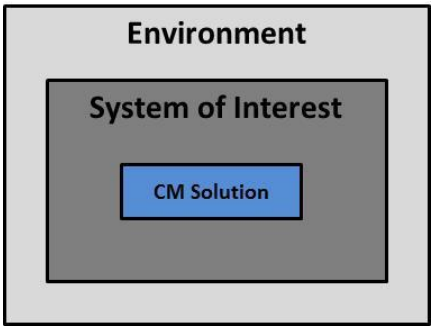

Figure 1.1: System of Interest, CM Solution, and Environment.

The DRIVER Test Bed provides tools to support experimentation of new DRIVER crisis management solutions. The Test Bed provides:

- Simulation tools that:
  - model the real world environment of the System of Interest, such as terrain, weather, infrastructure, flooding, earthquakes, crowds, groups, individuals, traffic, and media. These models are typed as "scenario presentation model" and "environment model".
  - model parts of the System of Interest itself, such as first responders, sensors, and command and control processes. These models are typed as "CM Actor/System model".
- Orchestration tools that:
  - orchestrate the execution of simulation tools,
  - facilitate the exchange of simulation data among simulation tools (conform the simulation data exchange model and simulation environment agreements),
  - facilitate the exchange of operational data between simulation tools and operational tools that are part of the System of Interest, and
  - collect simulation data or operational data for analysis.

The Test Bed models and tools are illustrated in Figure 1.2. Experimentation involves generally (amongst many other things) a scenario that defines the initial conditions and time line of significant events, and evidence that is collected by running the experiment.

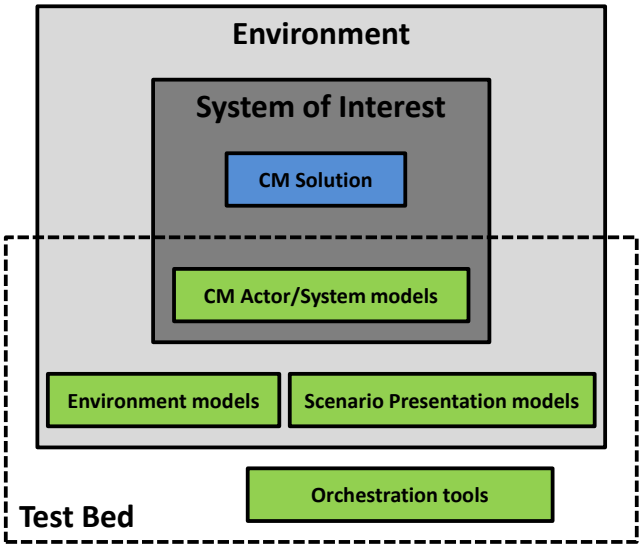

Figure 1.2: Classification of Test Bed models and tools.

1.4 Architecture development approach

The DRIVER Test Bed architecture description is developed in a number of iterations using a service oriented approach. This approach is based on the main steps in SOMA (Service-Oriented Modelling and Architecture), a methodology from IBM for the identification, specification and realization of services in a service oriented architecture [1]. The main three activities in this methodology are

Identify Services, Specify Services, and Design Services. These are illustrated in Figure 1.3 and further elaborated below.

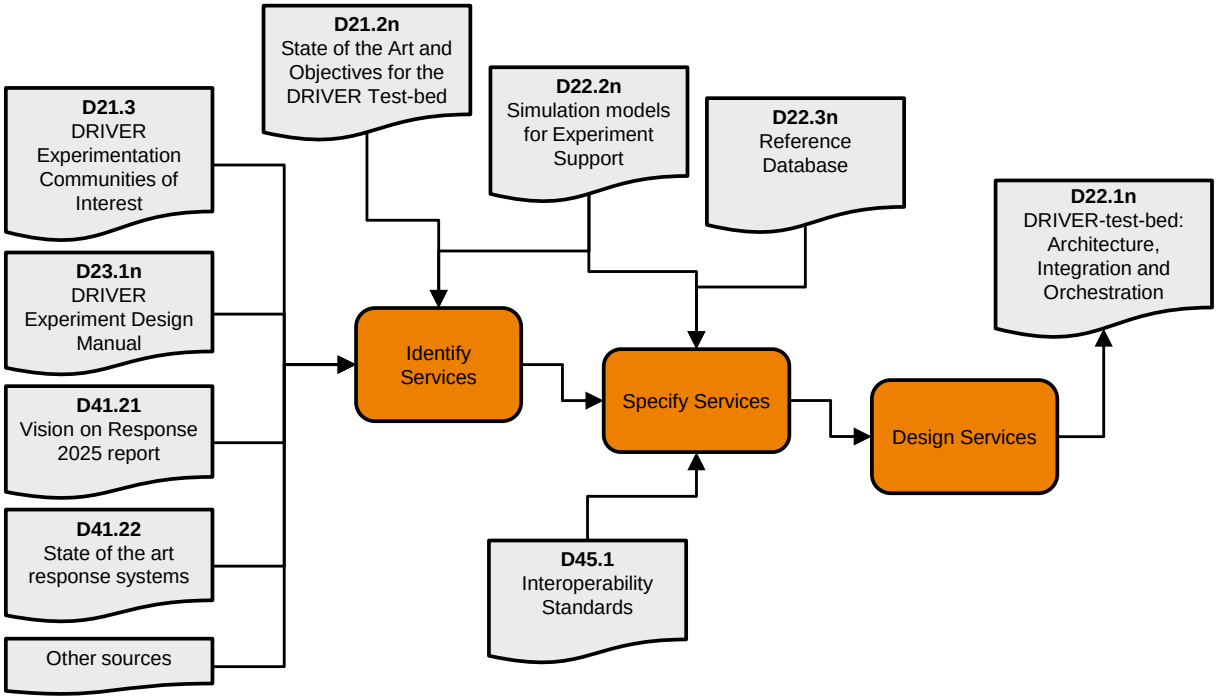

Figure 1.3: Service oriented approach: Identify, Specify, and Design Services.

The activities Identify Services, Specify Services, and Design Services are performed for each deliverable D22.1n (where n is 1..4), where the focus is on services identification in the initial deliverable (D22.11), and the focus shifts to services specification and services realization in later deliverables.

The deliverable D21.2n (where n is 1..3) is titled "*State of the Art and Objectives for the DRIVER Test-bed*" is one of the inputs to services identification. Similarly, the deliverable D22.2n (where n is 1..4) titled "*DRIVER-test-bed: Simulation models for Experiment Support*", and the deliverable D22.3n (where n is 1..4) titled "*DRIVER Reference Database*" are inputs to services specification. All these inputs converge to their final state over the course of the development of the DRIVER Test Bed architecture.

The above activities are performed by Task 22.2 partners, involving - as needed - other DRIVER partners (e.g. from SP4) utilizing the Test Bed in the architecture development discussions.

1.4.1 Identify services

The purpose of this activity is to identify candidate services that are involved in the Test Bed. Several techniques can be used for identifying services, such as goal-service modelling (using objectives from D21.2n), domain decomposition (a top down approach, using D21.3, D23.1n, D41.21), and existing asset analysis (a bottom up approach, using D21.2n, D41.22, and other relevant and further to be identified sources such as ACRIMAS).

Tasks within this activity include:

- Perform top-down analysis to identify services by using CM experimentation requirements;

- Perform bottom-up analysis to identify services, using the input deliverables;
- Perform goal-service modelling to identify services, by using the test bed objectives and performance measures of the CM experiments to identify required services;
- Categorize services in a service hierarchy or grouping;
- Allocate required functionalities to services.

1.4.2 Specify services

The purpose of this activity is to elaborate and detail the identified services, and to specify the service interfaces.

Tasks within this activity include:

- Specify service interfaces;
- Specify service dependencies on other services;
- Specify the flow of information among services;
- Develop service data exchange model;
- Develop service agreements;

1.4.3 Design services

The purpose of this activity is to evaluate service realization options and decide on which DRIVER Test Bed element will realize what service participant.

Tasks within this activity include:

- Analyse candidate DRIVER Test Bed elements for service realization;
- Evaluate service realization options;
- Determine technical feasibility;
- Record realization decisions;
- Document DRIVER Test Bed design;
- Design DRIVER Test Bed elements.

1.5 Document overview

The remainder of this document contains the following chapters:

Chapter 2 provides a description for each architecture viewpoint that is used in this document to describe the architecture of the DRIVER Test Bed.

Chapter 3 describes the architecture of the DRIVER Test Bed using the three viewpoints discussed in the previous chapter.

Chapter 4 provides rationales for architecture decisions made.

Chapter 5 provides conclusions.

The remaining parts of this document are the list of references and an annex. The annex contains information on terminology.

2 Architecture description

2.1 Architecture viewpoints

The DRIVER Test Bed architecture description is based on the concepts defined in [2], an international standard for the description of an architecture of a system. According to this standard the architecture of a system can be described using a variety of viewpoints that together form a set of views on the system under consideration and describe how the components of the system interact with each other.

Each architecture viewpoint defines several model kinds to represent aspects of the system. The name “model kind” refers to the conventions for a type of modelling, such as Unified Modelling Language (UML) activity diagrams for behavioural modelling. An architecture view expresses the architecture of a system from the perspective of an architecture viewpoint. An architecture view is composed of one or more architecture models that are constructed according to the conventions specified by the model kind governing each model.

The architecture description in this document uses the following viewpoints:

- Services viewpoint: the services viewpoint is used to describe the services that the DRIVER Test Bed generally should provide.
- System viewpoint: the system viewpoint is used to describe the DRIVER Test Bed architecture building blocks required to implement the desired services.
- Standards viewpoint: the standards viewpoint is used to specify the standards that may influence a DRIVER architecture building block. Standards are for example distributed simulation standards and simulation data exchange model standards.

2.2 Reference architecture and solution architecture

An architecture can be described at different levels of abstraction. Typically the name “reference architecture” is used for a more abstract form of architecture. A reference architecture generally provides a template solution for a concrete “solution architecture” and can be described using so called architecture building blocks. An architecture building block represents a component of the reference architecture that can be combined with other building blocks to deliver one or more solution architectures. A solution architecture can be viewed as an instantiation of a reference architecture, referring to concrete or physical realizations of the building blocks, such as certain orchestration tools and simulation tools. Some of these tools will be the same across solution architectures (typically the orchestration tools), others will be different per solution architecture (typically the simulation tools). The different concepts and their relationships are illustrated in Figure 2.1.

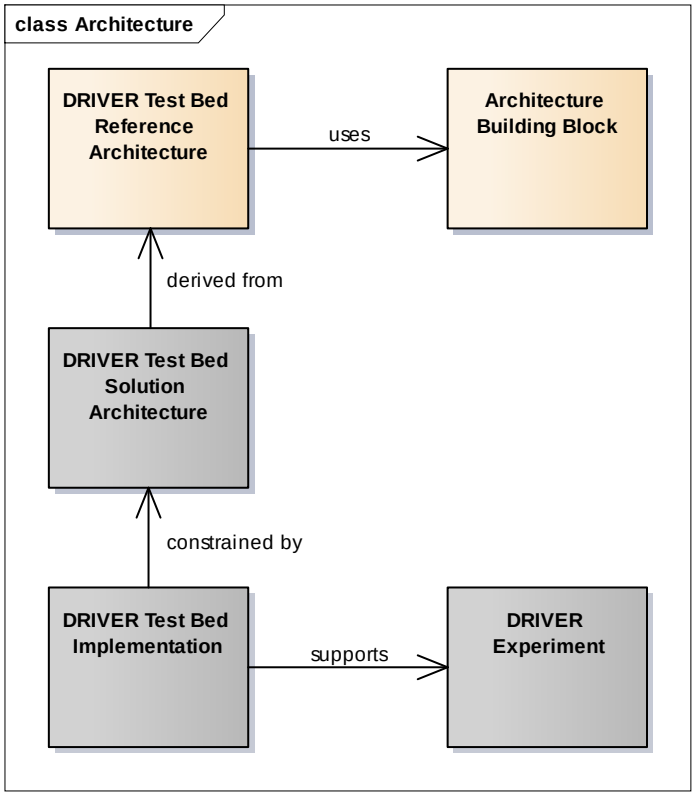

Figure 2.1: Reference architecture, architecture building block, and solution architecture.

In the context of DRIVER, this document (D22.1n) describes the DRIVER Test Bed reference architecture and the architecture building blocks. These elements are indicated with “sand” color in Figure 2.1; the “grey” colored elements are out of scope of this document.

The DRIVER Test Bed solution architecture is an instantiation of the DRIVER Test Bed reference architecture and addresses the organization of specific simulation and orchestration tools to support a certain DRIVER experiment. A list of available simulation and orchestration tools is provided in D22.2n (DRIVER-test-bed: Simulation models for Experiment Support). The Test Bed solution architecture(s) will be covered in D24.2x (The DRIVER Experimentation Support Tools).

The DRIVER Test Bed implementation is a concrete (and executable, software) implementation of the Test Bed that supports a certain experiment, as constrained by the DRIVER Test Bed solution architecture.

3 Test bed Architecture

This chapter describes the reference architecture of the DRIVER Test Bed using the three viewpoints discussed in the previous chapter.

3.1 Services view

The services view can be considered as a description of high level requirements, stating the services that the Test Bed must provide. An analysis of the input materials (see section 1.4) has yielded the following services considered relevant for the Test Bed. The Test Bed must, when it is required by an experiment, support the following services:

1. **Environment Representation**

   This service provides data on the natural environment, including permanent or semi-permanent man-made features. This includes data on terrain, weather, urban infrastructure such road information and buildings.

2. **Scenario Preparation**

   This service is used to prepare a scenario. This includes a service to suggest models that are able to participate in the test bed for the given scenario. This service uses the overview of models and related scenarios from D22.2n (DRIVER test-bed: Simulation models for Experiment Support) as a database for querying.

3. **Main Event List/Main Incident List Preparation**

   This service is used to plan events and incidents that are to be injected into the simulation execution during the execution step.

4. **Data Collection**

   This service collects data for after action review. This covers simulation data, live data, observer data, and execution management data. Observer data may be provided manually through a user interface. Live data includes voice, video, and messages from operational systems. For an overview of potential live data that are candidate for recording, see D45.1 (Interoperability Standards).

5. **Main Event List/Main Incident Scripting**

   This service scripts events and incidents that were previously planned during scenario preparation. The service also allows ad-hoc injection of events or incidents into the simulation execution.

6. **Visualisation**

   This services visualises ground truth and non-ground truth data.

   - 2D GIS map-based representation
   - 3D visual representation from multiple points of view (walk, fly (drone/helicopter/plane), drive, sail, birds-eye)

Note that the visualization service of the test bed may provide more than the Common Operational Picture (COP) of a CM tool. For example, ground truth data from simulation tools and events that happened.

7. **Simulation management**

   This service manages the simulation execution. Sub services are:

   1.7.1. Application management: service to start, stop, and monitor test bed elements
   1.7.2. Simulation initialization: service to provide initialization data to test bed elements
   1.7.3. Simulation control: service to monitor and control the simulation execution, and to manage simulation time (pause, resume, fast forward simulation, etc.).

8. **Simulation communication**

   This service enables simulation interoperability between member applications through a run time infrastructure[1], gateways and bridges, Simulation Data Exchange Model (SDEM)[2], and operating agreements. The SDEM describes the data that member applications can exchange at runtime.

9. **Operational communication**

   This service enables interoperability between simulation environment and operational systems, e.g. through gateways, data exchange models and operating agreements. For an overview of potential live data that are candidate for this service, see D45.1 (Interoperability Standards).

10. **Analysis**

    This service enables reviewing and analysis of previously recorded data.

11. **Time synchronization**

    Service to synchronize time across test bed elements (e.g. NTP service). This service is also available to participating tools.

12. **Networking services**
    - VPN service for secure communication across WAN(s)

13. **Database management**
    - To be determined, input from T22.3 (Reference Database)

14. **Test bed maintenance**
    - Save and restore configurations
    - Version management
    - Test-bed status monitoring
    - Issue tracking

15. **Test bed configuration**
    - Service to configure the test bed in order to establish a useful test configuration for the various experiments.

The identified Test Bed services can be aligned with the main process steps in experimentation (see [7]). A summary of the main process steps is provided in Table 3.1. Table 3.2 provides a cross

[1] As example, for HLA this corresponds to the HLA-RTI.
[2] As example, for HLA this corresponds to the FOM.

reference of service to most applicable process step. As can be concluded from this table, Test Bed services focus mostly on steps 3 to 5.

Experimentation steps	Description
1. Set goals and outcomes	The purpose of this step is to provide a clear formulation of the experiment, including a description of the problem to be addressed, the objectives to be reached and the propositions/hypotheses to be tested.
2. Select participants	The purpose of this step is to identify the participants in the experiment.
3. Prepare experiment	The purpose of this step is to develop the experiment and prepare for experiment execution.
4. Running the experiment	The purpose of this step is to conduct the experiment and collect the resulting data for analysis.
5. Interpret Evidence	The purpose of this step is to analyse and evaluate the data acquired during the experiment execution, derive conclusions and report the results.
6. Draw Conclusions	The purpose of this step is to draw meaningful conclusions.

Table 3.1: Main process steps in experimentation.

Experimentation steps	Set goals and outcomes	Select participants	Prepare experiment	Running the experiment	Interpret Evidence	Draw Conclusions
1. Environment Representation	X	X	X			
2. Scenario Preparation	X	X	X			
3. Main Event List/Main Incident List Preparation	X	X	X			
4. Data Collection			X	X	X	
5. Main Event List/Main Incident Scripting			X	X		
6. Visualisation			X	X	X	

Experimentation steps	Set goals and outcomes	Select participants	Prepare experiment	Running the experiment	Interpret Evidence	Draw Conclusions
7. Simulation management			X	X		
8. Simulation communication			X	X		
9. Operational communication			X	X		
10. Analysis				X	X	
11. Time synchronization			X	X		
12. Networking services			X	X		
13. Database management			X	X	X	
14. Test bed maintenance	X	X	X	X	X	X
15. Test bed configuration	X	X	X	X	X	X

Table 3.2: Cross reference of services to process steps.

3.2 Systems view

The systems view describes the architectural building blocks.

3.2.1 Test Bed as a federation of tools

As described earlier in chapter 2.2, there will be many implementations of the DRIVER Test Bed, each supporting one or more experiments. Each implementation has an architecture (called a solution architecture), which is derived from the reference architecture as described in this document. Every implementation of the Test Bed consists of several (simulation and orchestration) tools, in some way organized together to serve the purpose of the experiment. Some Test Bed tools interact with other Test Bed tools and are grouped together, some other Test Bed tools are used stand-alone, and yet other Test Bed tools interact with operational tools or end-users. In summary, many combinations of tools are possible to form what is called in literature a “Live, Virtual and Constructive” (LVC) simulation environment.

When Test Bed tools are integrated together in a group it is called a “federation” of tools. For tools to inter-operate in such a federation the data that tools may exchange must be defined, and so called

operating agreements must be established in order for tools to correctly operate as a whole. The data that can be exchanged between tools is defined in a so called “simulation data exchange model”, and the rules under which data is exchanged and tools interoperate are collectively called the “simulation environment agreements”.

In general, the DRIVER Test Bed is defined as a set of interoperating tools, where tools are organized in so called federations, along with a simulation data exchange model and set of operating agreements that are used as a whole to support the objective of the experiment. Some of these tools may be tightly coupled, i.e. exchange simulation data in a time-coherent manner. Other tools may be more loosely coupled or may be used independent of any other tool.

An example of a particular instantiation of the Test Bed Reference Architecture is shown in Figure 3.1. This figure shows an object model with three federations, each consisting of a number of tools. Colors are used to indicate the kind of tool. Light and dark grey colored tools are orchestration and simulation tools respectively. The green colored tools represent operational tools within the System of Interest, with which the Test Bed tools may interact. The connections between the tools indicate data exchange via some application programming interface. The data that can be exchanged amongst tools is described in a simulation data exchange model as will be discussed later.

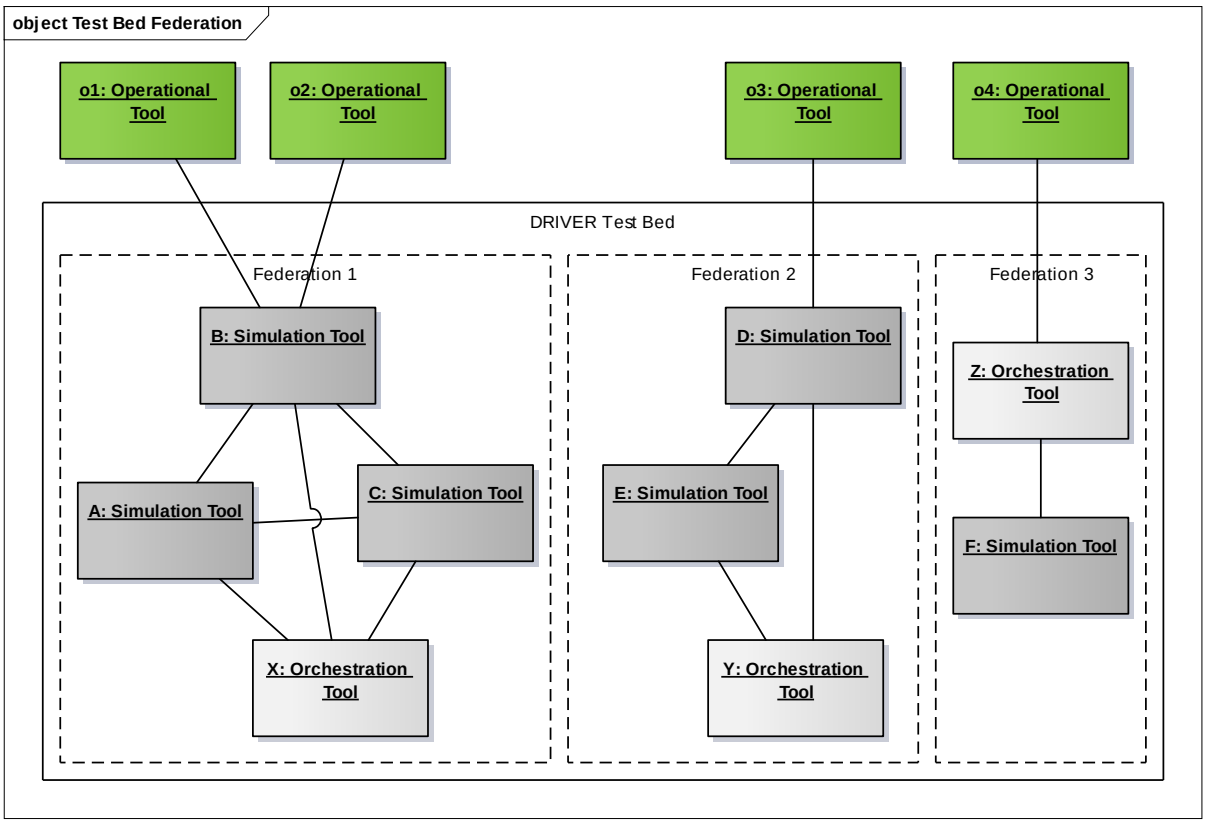

Figure 3.1: Example of a Test Bed instance.

The Test Bed is generally a federation of federations, where each federation consists of a set of tools. A small Test Bed may consist of just one federation with one simulation tool, whereas a large Test Bed may consists of many tools, partitioned in one or more federations.

3.2.2 Approach for coupling tools

Two common approaches for connecting simulation tools in a federation generally found in literature are:

- Pairwise coupling: every tool connects to every other tool as needed. For each connection a specific interface may need to be constructed, a dedicated data exchange model defined and operating agreements established. This approach may work fine for connecting just a few tools, but obviously when the number of tools grow also the number of connections grow rapidly! Furthermore, connections between tools may become solution specific, hampering on the end tool reuse.

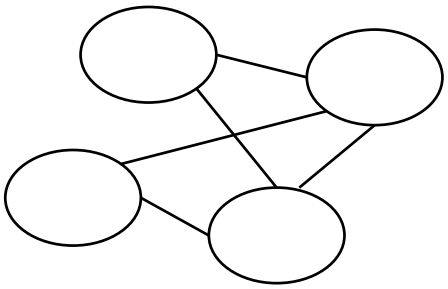

Figure 3.2: Pairwise coupling.

- Service bus coupling: in this approach each tool has a standard interface to a so called "service bus". This bus provides standard simulation services that tools may use to coordinate their activities, exchange data, and progress simulation time. Common topologies for a service bus are: centralized (communication between connected tools is via a central server component) or decentralized (communication is directly between connected tools), or a mix of these two. This approach has the advantage of limiting the number connections and interfaces and stimulating reuse of tools over time. Regardless of the topology, the tools use a common interface to communicate with each other. Often this common interface is realized by a software component called "middleware".

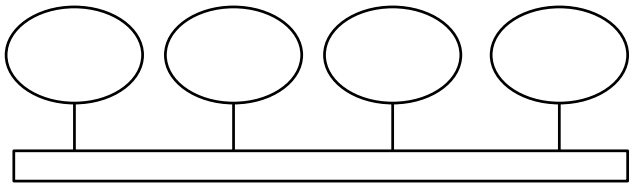

Figure 3.3: Service bus coupling.

The DRIVER Test Bed adapts the service bus coupling approach for connecting simulation tools. The coupling approach will use the High Level Architecture (HLA) as simulation architecture (see [8]). The HLA is a general reference architecture for distributed simulation and defines a service bus for connecting simulation tools (in HLA terminology tools are called "federates"). The service bus is called the HLA Run Time Infrastructure (HLA-RTI) and provides a number of services groups that are used by a federate to interact with the underlying communication layer:

1. Federation Management. These services allow for the coordination of federation-wide activities throughout the life of a federation execution. Such services include federation

execution creation and destruction, federate application joining and resigning, federation synchronization points, and save and restore operations.

2. Declaration Management. These services allow joined federates to specify the types of data they will supply to, or receive from, the federation execution. This process is done via a set of publication and subscription services along with some related services.
3. Object Management. These services support the life-cycle activities of the objects and interactions used by the joined federates of a federation execution. These services provide for registering and discovering object instances, updating and reflecting the instance attributes associated with these object instances, deleting or removing object instances as well as sending and receiving interactions and other related services. (Note: Formal definitions for each of these terms can be found in the definitions clause of all three HLA specifications.)
4. Ownership Management. These services are used to establish a specific joined federate's privilege to provide values for an object instance attribute as well as to facilitate dynamic transfer of this privilege (ownership) to other joined federates during a federation execution.
5. Time Management. These services allow joined federates to operate with a logical concept of time and to jointly maintain a distributed virtual clock. These services support discrete event simulations and assurance of causal ordering among events.
6. Data Distribution Management. These services allow joined federates to further specify the distribution conditions (beyond those provided via Declaration Management services) for the specific data they send or ask to receive during a federation execution. The RTI uses this information to route data from producers to consumers in a more tailored manner.
7. Support Services. This group includes miscellaneous services utilized by joined federates for performing such actions as name-to-handle and handle-to-name transformations, the setting of advisory switches, region manipulations, and RTI start-up and shutdown.

The HLA is not further elaborated in this document and the reader is referred to references [8], [9] and [10], and other references that can be found on the public internet.

3.2.3 Architecture building blocks

This chapter describes the architecture building blocks that can be used for constructing a Test Bed solution architecture and Test Bed implementation. Referring to [17] from the Open Group, an architecture building block represents a basic element of re-usable functionality, providing support for one or more capabilities, that can be realized by one or more components or products. The HLA is a reference architecture for connecting simulation tools in a federation of tools and architecture building blocks are used to provide specific focus on the logical aspects of the Test Bed reference architecture. An architecture building block will be instantiated as required to form a specific solution for a specific DRIVER experiment. A realization or instance of an architecture building block is called a solution building block, such as a certain software package.

The following table Table 3.3. provides an overview of the architecture building blocks, grouped by type of tool that the building block is applicable to.

Grouping	Architecture Building Block
Orchestration tool	Infrastructure Building Block
	Integration Building Block
	Monitoring and Control Building Block
Simulation tool	Scenario Presentation Model and Environment Model Building Block
	CM System/Actor model Building Block
Simulation Data Exchange Model	Ground truth SDEM Building Block
	Non-ground truth SDEM Building Block

Table 3.3: Architecture building blocks.

The following sections describe the architecture building blocks in more detail.

3.2.4 Orchestration tool

3.2.4.1 Infrastructure Building Block

Simulation Run Time Infrastructure

One of the most important architecture building blocks is the simulation Run Time Infrastructure. A simulation Run Time Infrastructure is generally an infrastructure that allows disparate tools to exchange simulation data. In general such an infrastructure provides software services for tools to coordinate their activities, data exchange and simulation time advancement. This building block will be realized by an HLA 1516-2010 compliant HLA-RTI implementation. Both commercial and open source (partial) HLA-RTI implementations are available. In HLA the tools that are connected by the run time infrastructure are called federates.

Time Server

A time server is an Infrastructure Building Block for synchronizing the system clocks in a computer network via NTP. Input and output data of tools is often time related. By using a synchronized clock it will be easier to correlate data from different tools. The Time Server will also be accessible by operational tools.

Note that there are generally two concepts of time w.r.t. a simulation: logical time and wall-clock time. The wall-clock time across different PCs should be synchronized via the Time Server. The logical time is whatever the tools within a certain Test Bed Implementation have agreed on, and is coordinated by the Simulation Run Time Infrastructure. Thus simulation tools that coordinate their time advancement outside of the simulation Run Time Infrastructure should use a synchronized wall-clock.

3.2.4.2 Integration Building Block

An Integration Building Block is used to connect simulation tools with operational tools, or to integrate legacy tools into the Test Bed.

Gateway and Adapter are two types of Integration Building Block. The names gateway, bridge and adapter are quite often used interchangeably. The word bridge is often misused in literature, sometimes referring to a gateway and sometimes to an adapter. In this document we use the following definitions:

- A gateway is an orchestration tool that is part of two architectures (for example, the HLA and the CIS) and translates data and services between both architectures. A gateway will be the typical solution for connecting simulation tools to the Common Information Space (CIS). A gateway that connects to the CIS is called “CIS Gateway”.
- An adapter is an orchestration tool that provides a set of software services that legacy tools or models can use to integrate with the simulation Run Time Infrastructure. An adapter will be the common solution for the integration of legacy tools with the Test Bed.

Figure 3.4 shows the structure of the CIS Gateway. The CIS Gateway is both an HLA federate that conforms to the Test Bed simulation environment agreements and a CIS application that conforms to the CIS agreements. It has the following components: a Local RTI Component (LRC) to communicate with the other federates within the federation, a CIS Connector to communicate with other applications within the CIS, and a mapper that maps data and services between the LRC and CIS Connector. The CIS Connector is a part of the CIS Adapter and is described in D42.1 (Final report on architecture design) and D42.21 (Specification documentation and deployment of the prototype and final integration platform). Note that there may be multiple CIS Gateway instances in a particular Test Bed implementation, each responsible for translating certain data defined in different FOM modules, such as Common Alerting Protocol (CAP) messages or Tactical Situation Object (TSO) data.

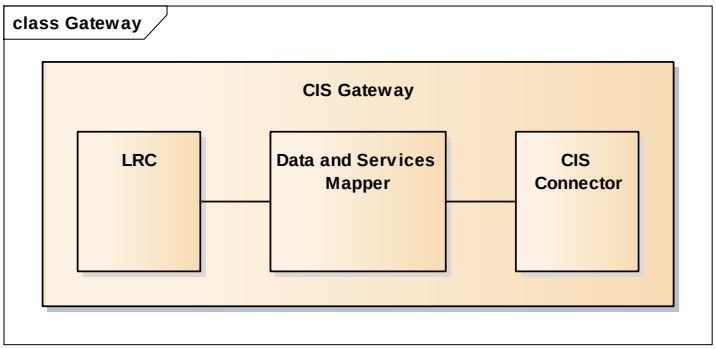

Figure 3.4: CIS Gateway.

A simulation tool may also communicate directly with the CIS. In that case the simulation tool includes a CIS connector. However, this variant is not an Integration Building Block, but a simulation tool type building block (see 3.2.6).

Figure 3.5 shows the structure of a simulation tool that uses an adapter to integrate with the simulation Run Time Infrastructure. An adapter typically provides a simple API dedicated to a particular model or class of models. An adapter may be FOM module specific (and generated by a code-generation tool) or may offer a subset of the LRC API to the simulation model. There are therefore different adapter instances, offering different APIs to simulation models.

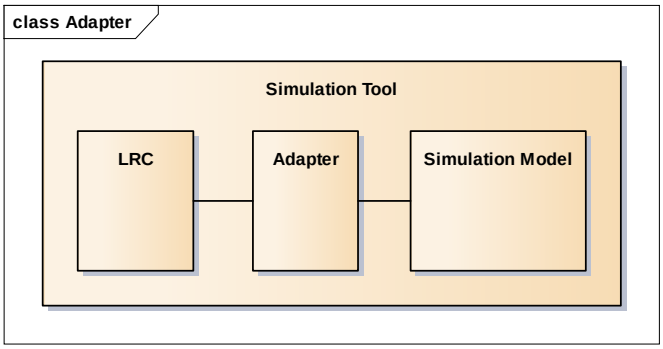

Figure 3.5: Adapter.

3.2.4.3 Monitoring and Control Building Block

The Monitoring and Control Building Block is a building block used to monitor and control Test Bed tools as well as to monitor and control the simulation performed by these tools. The following types of Monitoring and Control Building Block are distinguished: Controller, Data Viewer, and Data Recorder:

- Data Viewer: visualizes simulation data, often on a 2D map. Simulation data can be both ground truth data and non-ground truth data.
- Data Recorder: records simulation data, for later analysis.
- Application Controller: controls the state of Test Bed tools (e.g. start, stop), and
- Scenario Controller: controls the state of the simulation execution.

3.2.5 Simulation Data Exchange Model

3.2.5.1 SDEM Building Block

The Simulation Data Exchange Model (SDEM) is a specification of the information (a data model) that is exchanged run-time between Test Bed tools. For HLA the SDEM corresponds to the HLA Federation Object Model (HLA FOM). The HLA FOM describes amongst others the object classes, object class attributes, object class hierarchy, interaction classes and interaction class parameters for a simulation environment.

With the latest HLA standard (see [8]) logically related classes can be grouped in so called FOM modules, enabling component oriented development and stimulating the reuse of modules. The modularization of the FOM enables amongst others:

- Agreements related to a certain FOM module can be re-used between many federations;
- Extensions to a reference FOM can be put in a FOM module to avoid modifying standard FOMs;
- FOMs can become more agile as it easy to add a new or change an existing FOM module that only some federates use;
- A service oriented approach is possible where a federate defines the provided service data in a FOM module;

- A more decentralized approach with self-organizing federates can be applied: only federates that share the same FOM module exchange data and need to make agreements between each other.

An example of an SDEM with several modules is provided in Figure 3.6. Each module is depicted as a UML package and a dependency association is used where one module extends another module. A colour is used to indicate the kind of data that is modelled in the package: blue for ground truth data and green for non-ground truth data. As an example, some of the modules contain object classes.

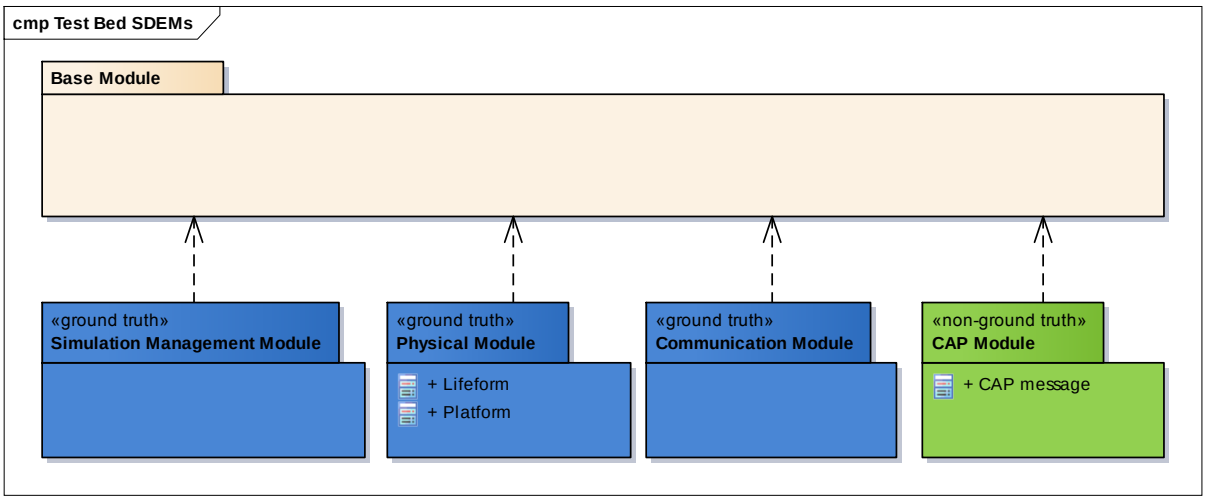

Figure 3.6: SDEM ground truth (blue) and non-ground truth (green) modules.

The SDEM schema defines the structure of an SDEM, similar to an XSD (a schema) for an XML file. For HLA there is a standard schema defined, called the Object Model Template (see [10]). This schema enables an object oriented and modular design of an HLA Federation Object Model. The advantage of using an SDEM schema is that it enables automation by tools, such as SDEM editing, code generation and run-time checks against an SDEM.

Although the SDEM represents an agreement among tools as to how runtime interaction will take place, there are other operating agreements that must be reached and that are not documented in the SDEM. Such agreements are necessary to establish a fully consistent, interoperable, simulation environment. There are many different types of agreements, for instance, agreements on initialization procedures for tools, synchronization points between tools, save/restore policies, progression of simulation time, object ownership, attribute update policies, security procedures, as well as algorithms that must be common across the Test Bed to achieve valid interactions among all tools. Many of these agreements will be however be solution specific and will not be described in this document.

A general structure for capturing (solution specific) agreements is the Federation Agreements Template (FEAT) (see [12]). The FEAT decomposes operating (federation) agreements in the following categories:

1. Metadata: Information about the federation agreements itself.

2. Design: Agreements about the basic purpose and design of the federation.
3. Execution: Technical and process agreements affecting execution of the federation.
4. Management: Systems/software engineering and project management agreements.
5. Data: Agreements about structure, values, and semantics of data to be exchanged during federation execution.
6. Infrastructure: Technical agreements about hardware, software, network protocols, and processes for implementing the infrastructure to support federation execution.
7. Modeling: Agreements to be implemented in simulation tools that semantically affect the current execution of the federation.
8. Variances: Exceptions to the federation agreements deemed necessary during integration and testing.

The FEAT is a template (XML schema) that can be leveraged to categorize simulation environment agreements (either using this schema directly, or using this schema to provide a structure for a textual format in an MS Word document).

Examples of simulation environment Execution agreements are:

- Execution states: agreements on federation execution states, e.g. initialization, saving, shutdown.
- Time management strategy: agreements on how simulation time will be advanced in the federation. That is: softRealTime, hardRealTime, scaledRealTime or asFastAsPossible. And per member application the strategy used, i.e. timeStepped, eventDriven, optimisticSynchronization, or paced with an external source.
- Update rates: agreements on how often member applications agree to update states; this may be static or dynamic. Might be upper/lower limits or set rates. Rates may be set for the entire federation, member applications, or object classes.

And examples of a Modeling agreements are:

- Effects adjudication: effects adjudication agreements ensure a 'fair fight' by specifying what component has the authority to determine the outcome or effect of an interaction between member applications, e.g. “shooter” (or the one initiating the effect) adjudicates, 'target' (the one the effect is perpetrated on) and, 'server' (some 3rd party member application).
- Coordinate systems: reference to authoritative coordinate system representations.

3.2.5.2 Ground truth and non-ground truth SDEM Building Block

The previous section described the SDEM and associated operating agreements as a general architecture Building Block. However, it is important to differentiate between two kinds of SDEM Building Block:

- Ground truth SDEM Building Block;
- Non-ground truth SDEM Building Block.

Where:

- Ground truth: The actual facts of a situation, without errors introduced by sensors or human perception and judgment.
- Non-ground truth: consisting of:
    - Perceived truth: data perceived by humans or sensors, including errors due to perception or judgment,
    - Other non-ground truth like orders or commands to persons or between systems.

The ground truth SDEM Building Block specifies ground truth data and can as such only be used to describe the data exchange between Test Bed tools (for example entity state data).

The non-ground truth SDEM Building Block specifies non-ground truth data. This can be data that is exchanged among Test Bed Tools, or between operational tools and Test Bed Tools via a gateway (for example CAP messages).

Ground truth data is data that resides inside the simulation as “the truth” and is used by simulation tools for modeling for example the environment of the SOI. This data is not exchanged with operational tools. Environment data that is used by operational tools such as terrain, number of inhabitants, location of critical infrastructure, hydrographic data, etc. should be considered as non-ground truth data. This is non-ground truth data because the number of inhabitants or locations of infrastructure may differ from the actual “truth” data used inside the simulation. Non-ground truth data can be exchanged between simulation tools and operational tools. Note however that in many instances the same data will be used for both ground truth and non-ground truth because there is no other data available. But in principle these are different and are modeled as such.

3.2.6 Simulation tool

The available simulation tools and models (Scenario Presentation Model, CM System/Actor model, and Environment Model) are described in D22.2n (DRIVER-test-bed: Simulation models for Experiment Support) and are not further elaborated here.

3.2.6.1 Scenario Presentation Model and Environment Model Building Block

This building block represents a simulation tool that models the real world environment of the System of Interest, such as terrain, weather, infrastructure, flooding, earthquakes, crowds, groups, individuals, traffic, and media. These models are typed as “scenario presentation model” and “environment model”.

3.2.6.2 CM System/Actor model Building Block

This building block represents a simulation tool that models parts of the System of Interest itself, such as first responders, sensors, and command and control processes. These models are typed as “CM Actor/System model”.

3.2.7 Services to architecture building block traceability

Architecture Building Block >	Infrastructure Building Block	Integration Building Block	Monitoring and Control Building Block	SDEM Building Block	Scenario Presentation Model and Environment Model Building Block	CM System/Actor model Building Block
1. Environment Representation					X	
2. Scenario Preparation					X	X
3. Main Event List/Main Incident List Preparation					X	X
4. Data Collection			X			
5. Main Event List/Main Incident Scripting					X	X
6. Visualisation			X			
7. Simulation management	X	X	X	X		
8. Simulation communication	X			X		
9. Operational communication	X			X		??
10. Analysis			X			
11. Time synchronization	X					
12. Networking services	X					
13. Database management			X			
14. Test bed maintenance						
15. Test bed configuration						

Table 3.4: Service to architecture building block traceability.

3.3 Standards view

This standards view lists the applicable standards. Applicable standards for the DRIVER Test Bed are provided in the table below.

Standard	Description	Service/ Component Name
CAP	Common Alerting Protocol See D45.1 (Interoperability Standards)	Simulation Data Exchange Model
EDXL	Emergency Data Exchange Language, in particular EDXL-DE as the container for EDXL messages See D45.1 (Interoperability Standards)	Simulation Data Exchange Model
IEEE 1516.1-2010	IEEE Standard for M&S High Level Architecture (HLA) – Federate Interface Specification	Simulation Run Time Infrastructure
IEEE 1516.2-2010	IEEE Standard for M&S High Level Architecture (HLA) – Object Model Template (OMT) Specification	Simulation Data Exchange Model
IEEE 1516-2010	IEEE Standard for M&S High Level Architecture (HLA) – Framework and rules	Test Bed tool
RFC 5905	Network Time Protocol Version 4: Protocol and Algorithms Specification	NTP Server
TSO	Tactical Situation Object See D45.1 (Interoperability Standards)	Simulation Data Exchange Model

Table 3.5: DRIVER test bed standards.

4 Architecture rationale

A decision in this deliverable is the selection of the HLA as simulation architecture for coupling Test Bed orchestration and simulation tools. Besides the HLA-RTI there are several other interoperability technologies that provide means for exchanging data between applications. It is possible to create simple data connections between applications using for example UDP/IP or TCP/IP, Simple Object Access Protocol (SOAP) or Representational State Transfer (REST). Or it is possible to create connections via message oriented middleware and protocols like the Advanced Message Queuing Protocol (AMQP) and the Data Distribution Service (DDS). From a low-level technical perspective this can always be seen as technology x can be replaced with technology y. Or that technology x can be implemented using technology y. As such, each of these technologies may provide a suitable solution when just simple services are needed, such as data transfer services. However, when connecting simulation tools additional, more advanced, simulation services might be needed. For example services for time management or services for ownership management. The HLA is a complete simulation architecture that provides basic data transfer services, as well as more advanced services used in simulations. Besides, several simulation tools used by DRIVER partners are HLA based.

Over the course of the DRIVER project several (orchestration and simulation) tools will be brought together in a Test Bed to support experimentation. Currently the HLA will be the single architecture of choice for connecting these tools. However, the need for complementary architectures will be closely monitored. Potentially resulting in a multi-architecture simulation environment.

5 Conclusion

This deliverable described the reference architecture of the Test Bed in terms of required services, architecture building blocks and applicable standards. The architecture is described using three architecture viewpoints (services viewpoint, systems viewpoint and standards viewpoint), each proving specific focus on aspects of the architecture.

There will be several implementations of the DRIVER Test Bed, each supporting one or more experiments. Each implementation has an architecture (called a solution architecture), which is derived from the Test Bed reference architecture as described in this document.

The HLA is a general reference architecture for distributed simulation and is selected as the reference architecture for connecting Test Bed tools. This document defined various architecture building blocks for constructing a Test Bed solution architecture and Test Bed implementation. Architecture building blocks include Infrastructure Building Block (e.g. HLA-RTI), Integration Building Block (e.g. gateway to operational tools), and Simulation Data Exchange Building Block (e.g. object model to describe the data that can be exchanged between Test Bed tools).

As further information will become available from other DRIVER deliverables and on-going experiments, the architecture description need to be updated. In particular the description needs to be updated with:

- Information on simulation data exchange models and simulation environment agreements;
    - Standard simulation data exchange models and agreements need to be defined to facilitate the re-use of Test Bed tools within different Test Bed implementations.
    - References to simulation and orchestration tools (in D22.2x) that support these data exchange models and agreements.
- Information on orchestration tools, in particular gateways and adaptors to operational tools;
    - Standard gateways and adaptors need to be defined to facilitate the coupling of Test Bed tools with operational tools.
- Information on environmental data and other data, such standards for terrain data, sensor data (Sensor Observation Service, SOS) and missing person data (People Finder Interchange Format, PFIF)
    - Standard formats need to be agreed on to facilitate the exchange of data between Test Bed tools.

References

[1] Norbert Bieberstein et al*, Executing SOA: A Practical Guide for the Service-Oriented Architect*, IBM Press, 2008.

[2] International Standard, *Systems and software engineering - Architecture description,* ISO/IEC 42010, IEEE Standard 42010-2011.

[3] Object Management Group, *Service Oriented Architecture Modeling Language (SoaML)*, Version 1.0.1, May 2012.

[4] Wikipedia, *SoaML*, http://en.wikipedia.org/wiki/SoaML.

[5] Object Management Group, *Systems Modeling Language (SysML)*, Version 1.3, June 2012.

[6] Wikipedia, *SysML*, http://en.wikipedia.org/wiki/Systems_Modeling_Language.

[7] D23.11 – DRIVER Experiment Design Manual.

[8] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules (IEEE 1516-2010).

[9] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification (IEEE 1516.1-2010).

[10] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (IEEE 1516.2-2010).

[11] IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP) (IEEE 1730-2010).

[12] SISO Federation Engineering Agreements Template (FEAT) Programmer's Reference Guide, http://www.sisostds.org/FEATProgrammersReference.

[13] SEDRIS Glossary, http://www.sedris.org/glossary.htm.

[14] Institute of Electrical and Electronics Engineers, *IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP)*, IEEE Standard 1730-2010.

[15] Institute of Electrical and Electronics Engineers, *Systems and software engineering - Architecture description*, IEEE Standard 42010-2011.

[16] International Standard, *Systems and software engineering - System life cycle processes*, ISO/IEC 15288, IEEE Standard 15288-2008.

[17] Open Group Standard, *SOA Reference Architecture* (C119), The Open Group, 2011.

[18] SISO draft standard, *Standard for Gateway Description Language*, SISO-STD-000-00-2014, 10 September 2014.

[19] DoD Modelling and Simulation (M&S) Glossary, December 2010.

Annexes

Annex A: Terminology

This annex defines the most relevant terms that are used throughout this document. Where possible authoritative sources are used in defining terms.

Term	Description
Architecture	Fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution. See [16].
Architecture building block	An Architecture building block represents a basic element of re-usable functionality, providing support for one or more capabilities, that can be realized by one or more components or products. See [17].
Bridge	Refers to a translator to link environments that use the same architecture. For example a bridge between two HLA federations. Based on definition in [18].
Environmental representation	An authoritative representation of all or part of the natural environment, including permanent or semi-permanent man-made features. See [13].
Gateway	Refers to a translator to link environments that use different architectures. Based on definition in [18]. Typically used between simulation environment and operational environment.

Ground truth data	The actual facts of a situation, without possible errors introduced by sensors or human perception and judgment. See [19]. As opposed to non-ground truth data.
Member application	An application that is serving some defined role within a simulation environment. This can include live, virtual, or constructive simulation assets, or can be supporting utility programs such as data loggers or visualization tools. See [14]. A member application may contain multiple simulation models. In the context of DRIVER a member application is either an orchestration tool or a simulation tool.
Model	A physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process. See [19].
Non-ground truth data	Data consisting of: • Perceived truth: data perceived by humans or sensors, including possible errors due to perception or judgment • Other non-ground truth like orders or commands to persons or between systems. As opposed to ground truth data.
Operational architecture	Architecture of the operational environment. See Architecture and Operational environment.
Operational data	Non-ground truth data exchanged between operational systems within an operational environment.
Operational environment	Collection of operational systems, used as a whole to achieve some objective.
Operational system	A real world system, such as a C4I system.
Operational tool	See operational system.

Orchestration tool	Is a member application that orchestrates the execution of simulation tools, facilitates the exchange of simulation data among simulation tools and between simulation tools and operational tools, and collects data for analysis. See also: member application.
Scenario	An initial set of condition and timeline of significant events imposed on trainees or systems to achieve exercise objectives. See [19].
Simulation data exchange model	A specification defining the information exchanged at runtime to achieve a given set of simulation objectives. This includes class relationships, data structures, parameters, and other relevant information. See [14].
Simulation architecture	Architecture of the simulation environment. See Architecture and Simulation environment.
Simulation data	Ground truth or non-ground truth data exchanged between member applications within a simulation environment.
Simulation environment	A named set of member applications along with a common simulation data exchange model (SDEM) and set of agreements that are used as a whole to achieve some specific objective. See [14].
Simulation tool	Is a member application that models the real world environment of the System of Interest and models parts of the System of Interest itself. See also: member application.
Test bed element	A member of a set of elements that constitutes the test bed. Based on definition in [16]. For example: member application, gateway, bridge, run time infrastructure.