CEN

WORKSHOP

AGREEMENT

CWA 17515

May 2020

ICS 03.100.01; 13.200; 35.240.99

English version

Building a common simulation space

This CEN Workshop Agreement has been drafted and approved by a Workshop of representatives of interested parties, the constitution of which is indicated in the foreword of this Workshop Agreement.

The formal process followed by the Workshop in the development of this Workshop Agreement has been endorsed by the National Members of CEN but neither the National Members of CEN nor the CEN-CENELEC Management Centre can be held accountable for the technical content of this CEN Workshop Agreement or possible conflicts with standards or legislation.

This CEN Workshop Agreement can in no way be held as being an official standard developed by CEN and its Members.

This CEN Workshop Agreement is publicly available as a reference document from the CEN Members National Standard Bodies.

CEN members are the national standards bodies of Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and United Kingdom.

EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

Ref. No.:CWA 17515:2020 E

Contents

European foreword

This CEN Workshop Agreement (CWA 17515:2020)has been developed in accordance with CEN-CENELEC Guide 29 'CEN/CENELEC Workshop Agreements – The way to rapid consensus' and with the relevant provision of CEN/CENELEC Internal Regulations – Part 2. It was approved by a Workshop of representatives of interested parties on 2019-07-09, the constitution of which was supported by CEN following the public call for participation made on 2019-06-10. However, this CEN Workshop Agreement does not necessarily reflect the views of all stakeholders that might have an interest in its subject matter.

This CEN Workshop Agreement (CWA) is based on the results of the DRIVER+ research project, which received funding from the European Union's 7th Framework Programme for Research, Technological Development and Demonstration under Grant Agreement (GA) N° 607798.

The final text of this CEN Workshop Agreement was submitted to CEN for publication on 2020-03-25.

The following organizations and individuals developed and approved this CEN Workshop Agreement:

— Independent Consultant/ Jean-François Sulzer;

— Nelen & Schuurmans B.V./ Govert ter Mors;

— Netherlands Organization for Applied Scientific Research/ Erik Vullings, Rinze Bruining;

— Riskaware Ltd./ Martyn Bull, Russell Mills, James England;

— Thales SIX GTS France SAS/ Jean-Benoit Bonne; and

— XVR Simulation B.V./ Martijn Hendriks, Steven van Campen.

Introduction

Due to the unimpeded population growth and our ever-increasing dependency on technology, effects in one domain can have a big impact on other domains, e.g. flooding can cause power and communication outages. Although the current focus on domain-specific simulators allows an increasing effectiveness of modeling a behavior to be as fine-grained as required, it also gives rise to the desire to interconnect multiple simulators together to simulate the co-dependence between different domains. Interoperability refers to the ability of computerized systems to connect and communicate with one another readily, even if they were developed by widely different manufacturers in different industries. CWA 17515 focuses on the interoperability between specialized simulators to facilitate a common simulation space used, for example, for exercises, operational setups and experiments that take place especially for crisis management inside the shared domain of these connected simulators.

Current and future challenges, due to increasingly severe consequences of natural disasters and terrorist threats, require the development and uptake of innovative solutions that address the operational needs of practitioners dealing with crisis management. DRIVER+ (Driving Innovation in Crisis Management for European Resilience) was a European research project funded under the 7th Framework Programme (FP7) that aimed to improve the way capability development and innovation management is tackled [1]. One of the objectives of this project was to develop an infrastructure to create relevant environments for enabling the trialing of new solutions and to explore and share crisis management capabilities. This development and use of the infrastructure within DRIVER+ provided a basis that led to the creation of this CWA.

For training and exercising in the crisis management domain, there is a growing need for a more realistic simulation of the crisis or incident, including an interactive 3D visualization of the incident site as well as the cascading effects that may follow the original incident. For example, a flooding may cause a power failure, which in turn causes a failure of the communication infrastructures and traffic management systems, leading to further disruptions. As these effects typically cover many domains, it is unlikely that there is a single simulator that covers all these aspects, so multiple simulators have to be connected and have to cooperate to create such an immersive fictive crisis.

Although similar standards regarding interoperability are available these standards are predominantly used within the military field. Examples for this are:

— IEEE 1516:2010 Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and rules

— IEEE 1516.1:2010 Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate interface specification

— IEEE 1516.2:2010 Modeling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (OMT) specification

— IEEE 1278.1:2012 Distributed Interactive Simulation (DIS) – Application protocols

— IEEE 1278.2:2015 Distributed Interactive Simulation (DIS) – Communication services and Profiles

— IEEE 1278.3:1996 Recommended practice for distributed interactive simulation – Exercise management and feedback

— IEEE 1278.4:1997 Recommended practice for distributed interactive simulation – Verification, validation, and accreditation

The implementation of such standards in the crisis management domain is difficult, because:

— tooling is expensive for crisis management organizations, there is no real open source alternative, and the community of practitioners is small;

— tooling typically only offers code generators for the programming languages Java, C++ and C#, and for applications using web service description language, which excludes many simulators;

— implementing such standards requires a steep learning curve, and although some work has been done in creating a crisis management Federation Object Model (FOM), no mature interaction standards are available;

— few, if any, simulators operating in the crisis management domain currently support HLA or DIS standards.

This CWA defines a set of architectural and interfacing guidelines for easily connecting simulators developed by different manufacturers from different industries. A reference implementation of the guidelines described in this document and created for DRIVER+ can be found at the DRIVER+ GitHub repository [2]. It has to be noted that other domains than crisis management may find benefit in implementing the dispositions of this CWA.

The structure of this document is as follows: Section 1 to 4 specifies general information about the content of the CWA, for example the scope, definitions and abbreviations. Section 5 defines architectural guidelines that are recommended when building a common simulation space, while Section 6 through 9 define the messages used inside the proposed architectural guidelines to communicate in the common simulation space. In order to provide the reader with practical examples of the use of the guidelines and messages, Annex B presents several use cases that represent common interactions when dealing with interoperability. These use cases intend to provide a frame of reference for the reader to understand the general vision of this CWA and for simulators to add specific protocols, messages, or applications relevant for their common simulation space in line with these guidelines.

1 Scope

This document defines a technical framework for connecting simulators and supporting tools aiming to facilitate interoperability between multiple stand-alone simulators, in order to jointly create and maintain a common simulation space. It specifies infrastructure and accompanied protocol parameters, common simulation message formats, and a set of services or tools facilitating the common simulation space functionalities. This document is intended to be used by system integrators and developers of individual simulators who jointly want to use an interoperability framework to share (parts of) their own simulation domain with simulators from another domain.

The aim for this CEN Workshop Agreement (CWA) is to provide a solid foundation of architectural guidelines to be used for jointly configuring a common simulation space. This CWA does not have the aim to closely integrate connected applications together. The general vision is that simulators are created for one or more specific domain knowledge areas with their own granularity, boundaries and purposes. To closely integrate these simulators would mean to integrate these domains as well, most likely causing irredeemable conflicts in the individual granularities, boundaries and purposes. In order to maintain individuality of the simulators, a common simulation space provides a framework for communication, based on a minimum commonality of the data accepted and produced by the individual simulators and an event-driven design philosophy.

The document provides a set of protocols and associated message formats to facilitate elementary interaction processes for simulators to function inside a common simulation space. To provide a better understanding of the proposed guidelines, this CWA also provides a repository of example interactions between simulators connected to a common simulation space. These examples are not described to limit the use of this document but are carefully chosen to reflect the most common types of interaction simulators would be expected to encounter when using a common simulation space. Each use case consists of a brief description of its intention, accompanied with a scenario description to provide an example for this use case. Based on this scenario, the desired information exchange flow and the required guidelines, messages, infrastructures and services to implement this flow are defined. Please note that the scenarios used inside each use case can be easily translated to other topics or configurations that serve the same purpose of the use case.

It is not the aim of this CWA to impose one global and general common simulation space to which all interested simulators have to connect to. A specific common simulation space should be configured based on, for example, the needs of an exercise, experiment or operational setup. This allows more flexibility for the simulators to find common ground for sharing domain-specific knowledge on a case-by-case basis. In order to ensure flexibility to fit the specific interoperability needs between simulators, the architectural guidelines posed in this document are categorized per interaction, allowing developers to jointly decide to implement the attached protocols and message formats, if required. If an interaction (e.g. change of ownership, updates of areas) is not necessary for the configured common simulation space or is not relevant for a specific simulator, its implementation is not mandatory.

Although numerous parties were involved in defining the architectural guidelines, it is inevitable that a specific (future) domain and/or interaction will be found that does not fit in the described architectural guidelines. Therefore, these guidelines focus on the elementary interaction processes, while supporting customization and expansion to fit specific interoperability needs. Because this CWA stresses the importance of a joint process of design and configuration, whenever the current architectural guidelines do not, it is encouraged to add new protocols and/or message formats that fit the need of a specific common simulation space.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at http://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1
applicant
connected application that is sending out a request

3.2
application
software or a program that is specific to the solution of an application problem

[SOURCE: ISO/IEC 20944-1:2013, 3.6.3.1]

3.3
common simulation space
virtual space aiming to facilitate interoperability between multiple stand-alone simulators

3.4
common simulation space infrastructure
underlying technical framework for the common simulation space

3.5
configuration guideline
procedure needed for further configuring a common simulation space for a specific purpose

3.6
external simulation space
virtual space with the same goal as the common simulation space that is not the common simulation space

3.7
message
data concept; grouping of data elements and/or data frames as well as associated message metadata, that is used to convey a complete unit of information

[SOURCE: ISO 24531:2013, 4.31]

3.8
request
message asking another connected application to perform a function

3.9
session
meeting of simulators working simultaneously according to the same protocol during a defined time period

[SOURCE: ISO 11136:2014, 3.14, modified: "consumers" from the original definition was exchanged with "simulators"]

3.10
session manager
connected application that enables a user to create, update and close a session, being responsible for informing the common simulation space regarding changes to the session

3.11
shared application space
virtual space aiming to facilitate interoperability between multiple applications other than simulators, consisting of protocols and messages tailored for a shared application domain

3.12
simulation domain
simulator-specific logically related group of components that provides the functions of that simulator

Note 1 to entry: The functions of that simulator are not shared on the common simulation space.

3.13
simulation entity
virtual object residing inside a simulation domain, consisting of a specific set of properties and functions

3.14
simulator
device, computer program, or system that behaves or operates like a given system when provided a set of controlled inputs

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.3750]

3.15
time manager
connected application that enables users to manage simulation time, being responsible for informing the common simulation space regarding changes of that simulation time

3.16
UNIX epoch time
amount of time that has elapsed since the UNIX epoch (00:00:00 UTC on 1 January 1970), ignoring leap seconds

3.17
virtual space
shared environment where applications from different domains exchange information syntactically and semantically in a meaningful way

4 Symbols and abbreviations

— API: Application programming interface

— CG: Configuration guideline

— CSS: Common simulation space

— DRIVER+: Driving innovation in crisis management for European resilience

— ECT: Exercise control tool

— ESS: External simulation space

— EXCON: Exercise control

— FOM: Federation object model

— OT: Observer/evaluator tool

— SAS: Shared application space

— ST: Storage tool

— URI: Uniform resource identifier

— UTC: Coordinated universal time

— G: Guideline

— sim: Simulator

— msg: Message

— HTTP: Hypertext transfer protocol

— WGS: World geodetic system

— CAP: Common alerting protocol

— ESMI: Emergency services messaging interface

5 Architectural guidelines

5.1 Prerequisites of a common simulation space infrastructure

In order to configure and use a common simulation space, several guidelines (G) regarding its infrastructure should be considered.

G.1 A common simulation space infrastructure should be based on well-developed and robust message broker software, where messages are exchanged between simulators and other tools.

This guideline implies that the main characteristics of a common simulation space infrastructure are reliant on the chosen message broker software. Although the core mechanisms of such software – like routing, subscription, policy, persistency, error handling, metadata and security – provide a solid basis for a common simulation space infrastructure, it also allows flexibility to choose the best fitting message broker for its specific needs.

G.2 Connecting to a common simulation space infrastructure should be made possible for all applications that are needed in the common simulation space, by means of configurable adapters that provide a documented application programming interface (API) for sending messages to and receiving messages from the specific common simulation space.

G.3 The common simulation space infrastructure should be modularized, having its core functionality of a message system expandable with additional modules for administrative functions, large data usage, data storage, application authentication and authorization, message encoding and encryption, as well as other features that might be needed for a specific common simulation space.

5.2 Configuration of a common simulation space

A general-purpose simulation space is not the intent of the provided guidelines; re-usability of protocols and message formats is. Each common simulation space supporting an exercise or training serves a specific purpose, requiring connected applications with different requirements, functionality and data exchange policies.

G.4 A common simulation space may exclude architectural guidelines defined inside this document. Since the guidelines inside this CWA are complementary within one sub-section, when excluding architectural guidelines, it is recommended to exclude the complete set of guidelines defined per sub-section instead of individual guidelines.

G.5 A common simulation space may include configuration guidelines for setting up that specific common simulation space.

G.6 Configuration guidelines should not conflict with the architectural guidelines included in this CWA but should serve as a specification on those guidelines.

G.7 A common simulation space may include additional message formats, i.e. not defined as part of this CWA, required to be sent over that specific common simulation space.

G.8 New common simulation space message formats should not semantically conflict with messages already defined inside this document.

The messages defined in this document are generic. The aim is to use these messages whenever possible to reduce the implementation work for the connected applications. New types of messages can be added, if they are needed for a specific common simulation space.

G.9 When configuring a common simulation space, the following should be defined:

- used common simulation space infrastructure;
- connected applications;
- used architectural guidelines;
- used configuration guidelines;
- used messages, including the properties inside each message;
- used pre-computed data or application-specific configurations;
- interactions defining which messages are sent and received by which connected application;
- common simulation space infrastructure channel or topic topology for facilitating the interactions;

— rules for conflict handling that might arise during interactions or handling of messages.

G.10 Connected applications should implement a translation between their own domain model and the messages they should send or receive inside the specific common simulation space.

EXAMPLE A simulator that has a location defined inside its local simulation space should implement a mapping between its space and the WGS 84 based space defined in a common simulation space location (see Table 19) in order to send out location information to the common simulation space.

5.3 Recognition of connected applications

Each connected application should notify all other connected applications that it is responsive for messages being sent over the common simulation space.

G.11 All connected applications should send out a heartbeat message (see Section 6.1) to the common simulation space, at a constant time interval agreed-upon in the configuration guidelines, to indicate the application is responsive in the common simulation space.

G.12 Each connected application interested in another connected application's responsiveness may listen to the heartbeat messages of that connected application.

With these guidelines, connected applications can infer the responsiveness of other connected applications. Especially for management or support tools (e.g. administration, session or time control), these guidelines can be crucial in understanding how to act inside the specific common simulation space.

Whenever the timespan between the current time and the timestamp reported in the latest received heartbeat message exceeds the agreed-upon time interval, e.g. two heartbeats, the application is deemed unresponsive until a new heartbeat is received.

5.4 Management of sessions

Whenever synchronization between multiple connected applications is needed, session management overarching these applications becomes relevant within a common simulation space. The most basic approach to session management makes sure all connected applications understand what state the current session is in.

G.13 Session management should be handled by one session manager.

This guideline does not enforce a connected application to solely function as a session manager. A connected application that serves as a session manager might also have another functionality – such as being a simulator or exercise control tool. There should, however, only one session manager be appointed for managing the session.

G.14 The session manager should inform the common simulation space of the current session information via the session management message (see Section 6.2).

G.15 Each connected application interested in session-related information should listen to the session management messages of the session manager.

G.16 Each session state change should be under supervision of a person or staff responsible for the session, making sure all connected applications are ready for the state change.

Guideline 16 allows, for example, for connected applications to prepare themselves for the actual start of the session during an initialization state, loading in pre-computed data or starting relevant functions defined statically in the configuration guidelines or dynamically in the session management message. Because this preparation time heavily relies on the individual connected application, personnel responsible for the session should make sure all connected applications are ready for the next state change before applying it via the session manager. This could be added to the human preparation phase of the session, checking if all operators or observers of the connected applications are ready to start the session.

5.5 Handling of time

Time management can become a crucial aspect of a common simulation space. Whenever more complex synchronization between connected applications is desired, different options are available that can only best be described in configuration guidelines. There is such a variety of handling time and timing conflicts per simulator that each common simulation space might have its own configuration regarding time management, and here only one approach is described in more detail.

G.17 Time management should be handled by a single time manager.

This guideline does not enforce a connected application to solely function as a time manager. A connected application that serves as a time manager might also have another functionality – such as being a simulator or exercise control tool – or the connected application is both session and time manager whenever both guideline sections are included in the specific common simulation space. There should, however, only a single time manager be appointed for managing the time inside one common simulation space.

G.18 Further definitions of time inside the common simulation space should be in the configuration guidelines.

> EXAMPLE The time manager can, for instance, be expanded to first synchronize time with all interested connected applications, to mitigate any system-specific time conflicts (e.g. time zones, operating system formats). An alternative to this would be to introduce a simulation time which is completely independent of any current time format stated by underlying systems (e.g. mark the start of the session as 0 and start counting seconds from that point onwards).

A simple form of time management is for the time manager to send out time information messages at an agreed-upon fixed time interval, and when intermediate changes occur, to send this information immediately as an event.

G.19 The time manager should inform the common simulation space of the current time information via the time management message (see Section 6.3) at a fixed time interval defined in the configuration guidelines.

G.20 The time manager should provide the current time and speed with each time management message.

G.21 Each connected application interested in time-related information based on intervals, should listen to the time management messages of the time manager.

G.22 Connected applications receiving a time management message should use the provided simulation time and speed for correcting their own application time mechanism to the provided time.

G.23 Whenever the time manager has changed its time state (e.g. the simulation is paused), it should inform the common simulation space of the current time information via the time management message (see Section 6.3).

G.24 Each connected application interested in time-related information based on time states, should listen to the time management messages of the time manager.

With these guidelines, the time manager described is expanded with logic to keep track of a pre-defined time interval. Although the user is still able to change the time states, the time manager sends out time management messages at a constant time interval. This allows connected applications that are relying on time to synchronize their simulation time with the leading common simulation space time.

5.6 Entities

A large part of a common simulation space is concerned with sharing data amongst multiple simulators, each with their own simulation domain. Simulation entity messages let simulators understand and update each other on data changes, without having to completely understand the domain specifics of all other simulators. These messages provide updates to shared entities inside a common simulation space, each with their own distinct properties (see Section 7).

G.25 An entity message should have a unique identifiable name (named id), to be used for referral by all connected applications.

G.26 An entity message may have an optional timestamp property, to represent the moment in time the update happened.

The time to use for this timestamp should be defined inside configuration guidelines, considering the time definition used inside Section 5.5 when applicable.

G.27 An entity message may have an optional type property, allowing the connected application sending the message to provide further information on what type of simulation entity it is sharing.

EXAMPLE 1 The type property might be an ambulance/firefighting vehicle specified further by car model, or a military/police officer specified further by rank. This type property allows for simulators to define and use a mapping for all shared entities inside a given common simulation space. For instance, all entities of type "subway station" can be displayed using simple icons and a name for one simulator, whereas another simulator may show them using realistic 3D models.

G.28 An entity message may have an optional tags property consisting of a list of unique string keys with accompanied string values, allowing the connected application sending the message to provide specific details that are relevant for the current common simulation space configuration.

Because of the generic applicability of the defined entity messages in Section 7, it is in the interest of the specific common simulation space to properly define the usage of the entity message properties.

EXAMPLE 2 The detailed specification of the tags property inside each entity message can vary. One common simulation space might require separating items of type person further into victims, crew and bystanders (by having "role" as key and "victim", "crew" and "bystander" as possible value), while another specifies tags for additional information concerning an area (e.g. defining a "water depth" as key with a number in meters as the value).

The tags property has been defined as a simple string-to-string mapping. Although more primitive and complex types can be defined for this, specifying a range of different types for a value makes the

recognition of which type is used more complex. A string-to-string mapping is understandable for all programming languages and reduces mapping complexity in translating the tags property back to useful information.

G.29 Whenever a connected application has created a new simulation entity that needs to be shared or the application has performed an update on a shared entity, the connected application should send the corresponding entity message with all relevant information to the common simulation space.

G.30 Unless configuration guidelines define otherwise, whenever a shared entity is no longer to be shared on the common simulation space, that entity can become marked as deleted by means of an entity deletion message (see Table 16).

Depending on the needs of a specific common simulation space, the last guideline can have many alternatives that should be defined in the configuration guidelines. An example of such an alternative is to include a specification of the tags property, adding a deleted state to report an entity needs to be unshared.

5.7 Aggregation of entity data

Whenever a subset of properties of a large amount of entities needs to be updated, aggregated messages can be used to limit the number of excess information being sent. Section 9.5 defines such an aggregation message, which can be used for wrapping all entity messages defined in Section 7.

G.31 Aggregation messages should only be used to provide updates to already shared entities.

G.32 Each defined entity message should also have its aggregated version containing a map with unique identifiers of the updated entities as a key, and entity aggregation information containing a subset of properties of the entity that can be changed throughout a session as a value.

G.33 All properties of entity aggregation messages should be marked as optional, to allow connected applications to send an update of a subset of these properties.

5.8 Requests

Where entity messages can be seen as notifications from one connected application to the other connected applications, request messages are asking other connected applications for changes or information outside the area of influence of the applicant (see Section 8).

G.34 A request message should have a unique identifiable name (named id), to be used for referral by all connected applications.

G.35 A request message should have an applicant property containing the identifier of the applicant.

G.36 A request message may have an optional tag property consisting of a list of unique string keys with accompanied string values, allowing the applicant to provide specific details that are relevant for the current common simulation space configuration.

G.37 Unless configuration guidelines define otherwise, any connected application may send out a request used in the common simulation space.

G.38 Unless configuration guidelines define otherwise, whenever a request is no longer applicable in a common simulation space, it may be cancelled by the applicant via a request cancellation message (see Table 17).

G.39 Each connected application that was handling the request and that received a request cancellation message concerning this request should, unless configuration guidelines define otherwise, stop current handling and leave the affected entities as is.

Additional configuration guidelines might, for instance, indicate reverting all affected entities to the state prior to starting the request.

5.9 Responding to requests

Depending on the specific common simulation space, responses may not be needed because a "fire and forget" policy is defined in the configuration guidelines, or responses can be inferred from future (in)actions of the connected applications receiving a request. Since the requirement of how to handle requests is common simulation space specific, a separation is created in the guidelines between the use of requests (see Section 5.8) and the use of responding to requests (Section 5.9).

Whenever more information or complex interactions are required inside a common simulation space, responses can be used to inform the applicant on the handling of its request.

G.40 A response message should have a unique identifiable name (named id), to be used for referral by all connected applications.

G.41 A response message should have a request property containing the identifier of the request this message is responding to.

G.42 Configuration guidelines should clearly define further implications of the possible responses for all connected applications in the common simulation space.

Understanding what it means to have an approval, rejection or error response from a connected application is important in handling interactions inside a common simulation space properly, and this can only be correctly defined inside the configuration guidelines.

5.10 Transfer of ownership

Within a common simulation space, transferring ownership allows connected applications to become owners of a shared entity and have the exclusive right to change properties of that shared entity. When interaction complexity inside a common simulation space increases, control of ownership may be crucial to provide clear restrictions on when to apply updates to a shared entity. This can happen in situations where multiple connected applications update the same shared entity over time based on specific application functionality.

G.43 An entity message should have an optional owner property (only when the entity can be shared) to represent the current owner of this simulation entity.

G.44 Whenever a connected application creates an entity that can have multiple owners (but only one at a time) inside a common simulation space, the owner property of that entity should be set to the identifier of the connected application, marking this connected application as owner.

G.45 Connected applications should only send (aggregated) entity messages for entities that they are owner of.

G.46 Whenever a connected application wants to own a simulation entity that it does not own, the connected application should send a request for ownership (see Section 8.1).

G.47 Unless configuration guidelines define otherwise, an owner receiving a request for ownership of a simulation entity it owns, should send out an entity message with its owner property changed to the applicant (thereby handing over the ownership to that applicant).

Additional configuration guidelines may be needed to prevent conflicts in transferring ownership (e.g. two simulators constantly swapping ownership of the same entity without any progress, or a continuous stream of ownership requests for any entity that will never be approved by its current owner).

An alternative to transferring ownership as defined in G.46 and G.47 could be to introduce specific request messages (possibly re-using the aggregation message properties) for the connected applications to implement. Instead of requesting ownership of an entity to change properties by itself, the applicant then requests the owner to change those properties on behalf of the applicant.

These guidelines describe ownership at a simulation entity level, limiting the number of owners per entity to one. One might deem it feasible to define ownership at the property level of those entities, allowing entities to have multiple owners at the same time, or to implement a more complex service for facilitating this behavior. In order to achieve this, several messages defined in this document need to be altered. It also introduces additional configuration guidelines to make sure no conflicts can arise inside a common simulation space. Since this CWA focuses on defining a basis for building a common simulation space, it advises on using the ownership guidelines only at simulation entity level.

6 Configuration messages

6.1 Heartbeat message

A heartbeat message is used to derive the state of responsiveness of a connected application, further defined in Section 5.3 (see Table 1).

Table 1 — Heartbeat message properties

Property	Type	Description
id	string	Unique case-insensitive identifier of the connected application sending out the heartbeat message.
alive	long	UNIX Epoch time in milliseconds marking the time the connected application last responded.
origin	string	Optional case-insensitive identifier from which device or location the heartbeat message was sent. EXAMPLE IP address or machine name.

6.2 Session management message

The session management message is used for informing connected applications on session details and primarily the current state of the session (see Table 2). Section 5.4 defines guidelines that use this session management message.

Table 2 — Session management message properties

Property	Type	Description
id	string	Unique case-insensitive identifier of the session.
state	enum	State the session is currently in: *Initializing* – preparing for the actual start of a session, including loading in pre-computed data and configuring the connected application as was designed for this particular use of the common simulation space *Started* – the session has started and is running *Stopped* – the session has stopped *Closed* – the session is closed after all administrative tasks are done, so no more messages can be exchanged
name	string	Optional name of the session.
tags	map<string>	Optional map containing session-specific information: *key* – unique name of the specific property *value* – value of that property
timestamp	long	Optional UNIX Epoch time in milliseconds marking the time the update was or needs to be performed.
simulationTime	long	Optional UNIX Epoch time in milliseconds marking the fictive date and time the session should run with.

6.3 Time management message

The time management message can be used for informing connected applications on time progression and changes (see Table 3). Section 5.5 defines guidelines that show options of using this time management message.

Table 3 — Time management message properties

Property	Type	Description
state	enum	State the time is currently in: *Initialization* – preparing for the actual start of the simulation time *Started* – the simulation time is started *Paused* – the simulation time is paused *Stopped* – the simulation time is stopped *Reset* – the simulation time is reset
tags	map<string>	Optional map containing time specific information: *key* – unique name of the specific property

		value – value of that property
timestamp	long	Optional UNIX Epoch time in milliseconds marking the time the update was or needs to be performed.
simulationTime	long	Optional UNIX Epoch time in milliseconds marking the fictive date and time the session should run with.
simulationSpeed	float	Optional speed factor this session wants to run a simulation. Range of the speed factor: [0, ∞).

7 Entity messages

7.1 Item message

An item is a specific simulation entity that is bound to one position in the world (see Table 4). It should represent a tangible object, person or vehicle.

Items can have a parent-child relation described in the parent entity by means of a list of children, e.g. an ambulance vehicle has two paramedics as children. Upon receiving this item message, the children may or may not be known to the receiving simulator, which is something that should be specified in the configuration guidelines. More complex hierarchies can be constructed in this way too.

Table 4 — Item message properties

Property	Type	Description
id	string	Unique case-insensitive identifier of the item.
location	Location	Location of the item (see Table 19).
orientation	Orientation	Optional orientation of the item (see Table 20).
velocity	Velocity	Optional velocity of the item (see Table 21).
name	string	Optional name of the item.
description	string	Optional description of the item.
type	string	Optional type of the item.
owner	string	Optional unique case-insensitive identifier of the connected application owning the item.
timestamp	long	Optional UNIX Epoch time in milliseconds marking the time the update was performed.
tags	map<string>	Optional map containing item specific information: *key* – unique name of the specific property *value* – value of that property
children	list<string>	Optional list of item identifiers that belong to this item.

7.2 Feature collection message

A feature collection represents one or more points, lines, or areas of interest to the common simulation space. For example, specific places or buildings, routes, borders, flooded areas or disaster zones. A specification for sharing a collection of geographic features is GeoJSON (RFC 7946:2016), created by the Internet Engineering Task Force. Each feature can be a point, multi-point, line, multi-line, polygon, or multi-polygon (with or without holes). In addition, every feature has a map of arbitrary properties for other standards or specifications to include relevant information per feature. One of those specifications this CWA recommends in using is the simplestyle specification proposed by MapBox [7], which defines properties for visual styling a feature's geometry.

Similar to the simplestyle specification, this CWA defines a list of common feature properties for structuring additional information about its context on top of the GeoJSON specification, making it easier to share simulation relevant data (see Table 5).

A feature might have a list of entities that are present at that specific feature. This can be used, for instance, to indicate the fire truck units and materials that are stationed at a fire station. In order to reduce the amount of duplicate information that can be sent over, this list only contains the unique case-insensitive identifiers of the shared entities that reside at that feature.

Table 5 — Feature collection message properties

Property	Type	Description
id	string	Unique case-insensitive identifier of the feature collection.
type	enum	Type of the feature collection (as defined by the GeoJSON specification): *FeatureCollection* – a collection of multiple GeoJSON features *Feature*– a single GeoJSON feature (not used within this CWA) *Geometry* – a single geometric GeoJSON object (not used within this CWA) In this CWA only the *FeatureCollection* option is used for easier processing.
features	list<Feature>	List of all features inside the feature collection (as defined by the GeoJSON specification) (see Table 6).
name	string	Optional name of the feature collection.
description	string	Optional description of the feature collection.
owner	string	Optional unique case-insensitive identifier of the connected application owning the feature collection.
timestamp	long	Optional UNIX Epoch time in milliseconds marking the time the update was performed.
tags	map<string>	Optional map containing feature collection specific information: *key* – unique name of the specific property

		value – value of that property
bbox	list<double>	Optional bounding box around the feature collection in the following order [west, south, east, north]. — $length(bbox) = 2 \times n$, — where *n* is the number of dimensions represented in the contained geometries, with all axes of the most south-westerly point followed by all axes of the more north-easterly point. The axes order of a bbox follows the axes order of the geometries. The bbox values define shapes with edges that follow lines of constant longitude, latitude, and elevation.

Table 6 — Feature properties

Property	**Type**	**Description**
type	enum	Type of the feature (as defined by the GeoJSON specification): *Feature* – a single GeoJSON feature
geometry	Geometry	Geometry of the feature (as defined by the GeoJSON specification) (see Table 7).
properties	Properties	Optional feature specific properties (as defined by the GeoJSON specification) (see Table 8).
bbox	list<double>	Optional bounding box around the feature in the following order [west, south, east, north]. — $length(bbox) = 2 \times n$, — where *n* is the number of dimensions represented in the contained geometry, with all axes of the most south-westerly point followed by all axes of the more north-easterly point. The axes order of a bbox follows the axes order of the geometry. The bbox values define shapes with edges that follow lines of constant longitude, latitude, and elevation.

Table 7 — Geometry properties

Property	Type	Description
type	enum	Type of the geometry (as defined by the GeoJSON specification): *Point* – a point *MultiPoint* – a collection of points *LineString* – a collection of points forming a line *MultiLineString* – a collection of lines *Polygon* – a collection of points forming an area *MultiPolygon* – a collection of areas *GeometryCollection* – a collection of any of the types above
coordinates	list<double> or list<list<double>> or list<list<list<double>>>	List of coordinates representing the geometry (as defined by the GeoJSON specification). For each coordinate, the first two elements are longitude and latitude. Altitude can be included as an optional third element. Longitude, latitude and altitude are further specified in Table 19. The difference in coordinate types depends on the chosen geometry type: a point uses a single coordinate (a list of decimals), while a multi polygon uses a list of polygons, each having a list of coordinates, in its turn represented by a list of decimals. Polygon rings follow the right-hand rule for orientation (counterclockwise order for external rings, clockwise order for internal rings).

Table 8 — Property properties

Property	Type	Description
id	string	Case-insensitive unique identifier of the feature.
name	string	Optional name of the feature.
description	string	Optional description of the feature.
type	string	Optional type of the feature.
tags	map<string>	Optional map containing feature specific information: *key* – unique name of the specific property *value* – value of that property
orientation	Orientation	Optional orientation of the feature (see Table 20).
entities	list<string>	Optional list of entity identifiers that are at the feature.
address	Address	Optional address information of the feature (see Table 9).

Table 9 — Address properties

Property	Type	Description
street	string	Optional street name including house number and house letter.
houseNumber	int	Optional house number.
houseLetter	string	Optional house letter.
postalCode	string	Optional postal code.
city	string	Optional name of the city.
state	string	Optional name of the state or province.
country	string	Optional name of the country.
tags	map<string>	Optional map containing address specific information: *key* – unique name of the specific property *value* – value of that property

7.3 Post message

A post is a specific simulation entity representing a piece of writing, image or other content published. Examples of posts can range from (e-)mail to social media posts (see Table 10).

Table 10 — Post message properties

Property	Type	Description
id	string	Unique case-insensitive identifier of the post.
body	string	Body text of the post.
header	Header	Optional header information of the post (see Table 11).
name	string	Optional name of the post.
type	string	Optional type of the post.
owner	string	Optional unique case-insensitive identifier of the connected application owning the post.
timestamp	long	Optional UNIX Epoch time in milliseconds marking the time the update was performed.
tags	map<string>	Optional map containing post specific information: *key* – unique name of the specific property *value* – value of that property

Table 11 — Header properties

Property	Type	Description
from	string	Sender of the post.
date	long	UNIX Epoch time in milliseconds marking the time the post was published/updated.
to	list<string>	Optional list of recipients of the post.
cc	list<string>	Optional list of recipients in carbon copy of the post.
bcc	list<string>	Optional list of recipients in blind carbon copy of the post.
subject	string	Optional subject of the post.
intro	string	Optional introductory text of the post.
attachments	map<string>	Optional map of (references to) attachments inside the post: *key* – unique reference to the attachment (e.g. Uniform Resource Identifier (URI)) or complete string-encoded attachment *value* – media type of the attachment (e.g. .pdf, .png, .zip)
location	Location	Optional location of the sender of the post (see Table 19).

8 Request messages

8.1 Ownership request message

An ownership request is a specific request for becoming owner of a given simulation entity (see Table 12). Section 5.10 defines guidelines that use this ownership request message.

Table 12 — Ownership request message properties

Property	Type	Description
id	string	Unique case-insensitive identifier of the request.
applicant	string	Unique case-insensitive identifier of the connected application sending the request.
entity	string	Unique case-insensitive identifier of the entity the applicant requests ownership over.
tags	map<string>	Optional map containing ownership request specific information: *key* – unique name of the specific property *value* – value of that property

8.2 Route request message

A route request is a specific request for calculating a route from a starting location, passing through several waypoints (see Table 13).

Table 13 — Route request message properties

Property	Type	Description
id	string	Unique case-insensitive identifier of the request.
applicant	string	Unique case-insensitive identifier of the connected application sending the request.
start	Location	Location that marks the start of the route (see Table 19).
waypoints	list<Location>	Ordered list of locations to visit consecutively (see Table 19).
moveType	enum	Optional type of movement to consider calculating the route: *Straight* – move in a direct line to all waypoints without taking into account the terrain *CrossCountry* – move directly to all waypoints without taking into account the roads *OnlyRoads* – stay on the roads to get to the closest point to the waypoints that is still on a road *RoadsAndCrossCountry* – move to the waypoints by taking into account the roads; it is allowed to go off the road
tags	map<string>	Optional map containing route request specific information: *key* – unique name of the specific property *value* – value of that property

8.3 Move request message

A move request is a specific request for moving or transporting a given simulation entity towards a given destination, possibly over a given set of waypoints or route (see Table 14).

Table 14 — Move request message properties

Property	Type	Description
id	string	Unique case-insensitive identifier of the request.
applicant	string	Unique case-insensitive identifier of the connected application sending the request.
entities	list<string>	Unique identifiers of the entities the applicant requests to be moved. This also allows multiple entities to move in a convoy.

destination	string	Unique identifier of the entity the applicant requests the given entities to move to.
waypoints	list<Location>	Optional list of locations to visit consecutively along the movement (see Table 19).
moveType	enum	Optional type of movement: *Straight* – move in a direct line to all waypoints without taking into account the terrain *CrossCountry* – move directly to all waypoints without taking into account the roads *OnlyRoads* – stay on the roads to get to the closest point to the waypoints that is still on a road *RoadsAndCrossCountry* – move to the waypoints by taking into account the roads; it is allowed to go off the road
route	string	Optional unique identifier of a feature collection representing the preferred route that should be followed (see Table 5). This property should not be used together with the waypoints property.
tags	map<string>	Optional map containing move request specific information: *key* – unique name of the specific property *value* – value of that property

In principle, the owner of the entity should handle the request. If multiple connected applications are capable of handling move requests, configuration guidelines should be defined resolving all possible conflicts that might arise. An additional guideline could be to split movement per capable connected application based on entity type or specific tag combinations (e.g. move requests concerning vehicles marked as traffic are handled by a connected traffic simulator, while move requests concerning persons marked as pedestrian are handled by a connected crowd simulator).

8.4 Start inject request message

A start inject request is a specific request for starting a pre-defined sequence of events defined at one or more connected applications (see Table 15).

Table 15 — Start inject request message properties

Property	Type	Description
id	string	Unique case-insensitive identifier of the request.
applicant	string	Unique case-insensitive identifier of the connected application sending the request.
inject	string	Case-insensitive name of the inject that is requested to start.
tags	map<string>	Optional map containing start inject request specific information: *key* – unique name of the specific property *value* – value of that property

If multiple connected applications are capable of handling start inject requests, all of them should start the given inject if present in their implementation. This allows a common simulation space to have global injects that have application-specific effects, and eases starting of said injects without knowing all specific details per application.

9 Support messages

9.1 Entity deletion message

An entity deletion message is a support messages indicating a shared entity should no longer be shared (see Table 16).

Table 16 — Entity deletion message properties

Property	Type	Description
id	List<string>	Unique case-insensitive identifier of the entity.
owner	string	Optional unique case-insensitive identifier of the connected application owning the entity.
timestamp	long	Optional UNIX Epoch time in milliseconds marking the time the update was performed.

9.2 Request cancellation message

A request cancellation message is a support message indicating a pending request to be cancelled (see Table 17).

Table 17 — Request cancellation message properties

Property	Type	Description
id	string	Unique case-insensitive identifier of the request.
applicant	string	Unique case-insensitive identifier of the connected application sending the request.
timestamp	long	Optional UNIX Epoch time in milliseconds marking the time the update was performed.

9.3 Response message

A response message is a support message for responding to a received common simulation space message (see Table 18). Guidelines for responding to requests are described in Section 5.9.

Table 18 — Response message properties

Property	Type	Description
id	string	Unique case-insensitive identifier of the response.
request	string	Unique case-insensitive identifier of the request this response is responding to.
code	int	Optional HTTP status code [100] that best serves the response. Configuration guidelines might define new response codes that better fit the needs of that common simulation space.
message	string	Optional information accompanying the response code.
timestamp	long	Optional UNIX Epoch time in milliseconds marking the time the respond was given.

9.4 Geo-referencing

A location is defined as a WGS 84 based standard [3] representation of a location relative to the WGS 84 based ellipsoid (see Table 19).

Table 19 — Location properties

Property	Type	Description
latitude	double	In decimal degrees, ranging from [-90, 90] where 0 is the equator.
longitude	double	In decimal degrees, ranging from (-180, 180] where 0 is the prime meridian (line going through the geographic north, Greenwich, and the geographic south).
altitude	double	Optional in meters, where 0 is the surface of the WGS 84 based ellipsoid, or another agreed upon common ground level (specified inside the configuration guidelines). A positive number indicates a location outside the ellipsoid (or above the ground level), while a negative number indicates a location inside the ellipsoid (or below the ground level). If an altitude is not provided, it is presumed that the location is at the ground level of the provided latitude and longitude coordinates.

An orientation is defined in the aviation axes conventions representation (see Table 20). It is a left-handed item-centric reference system, with in default initial state its heading/yaw-axis pointing away from the center of the WGS 84 based ellipsoid, its pitch-axis pointing to the right, and its roll/bank-axis pointing forward. The orientation of an item is represented in three properties, each representing a rotation over one of the defined axes.

Table 20 — Orientation properties

Property	Type	Description
yaw	double	In decimal degrees, ranging from [0, 360) where 0 is pointing towards the geographic north. The yaw value is applied in a clockwise rotation over the item's heading/yaw-axis. A yaw value of 90 makes the item face east, while a yaw of 270 makes it face west.
pitch	double	In decimal degrees, ranging from [-90, 90] where 0 is perpendicular to the line crossing the item's location and the center of the WGS 84 based ellipsoid. The pitch value is applied in a counter-clockwise rotation over the item's pitch-axis. A pitch value of 45 makes the item face 45 degrees upwards, while a pitch of -20 makes it face 20 degrees downwards.
roll	double	In decimal degrees, ranging from (-180, 180] where 0 is perpendicular to the line crossing the item's location and the center of the WGS 84 based ellipsoid. The roll value is applied in a clockwise rotation over the item's roll/bank-axis. A roll value of 45 makes the item roll 45 degrees to the right, while a roll of -50 makes it roll 50 degrees to the left.

A velocity is defined in the aviation axes conventions representation of a velocity vector (see Table 21). It is a left-handed item-centric reference system, with in default initial state its heading/yaw-axis pointing away from the center of the WGS 84 based ellipsoid, its pitch-axis pointing to the right, and its roll/bank-axis pointing forward.

Table 21 — Velocity properties

Property	Type	Description
yaw	double	In decimal degrees, ranging from [0, 360) where 0 is pointing towards the geographic north. The yaw value is applied in a clockwise rotation over the item's heading/yaw-axis. A yaw value of 90 makes the item face east, while a yaw of 270 makes it face west.
pitch	double	In decimal degrees, ranging from [-90, 90] where 0 is perpendicular to the line crossing the item's location and the center of the WGS 84 based ellipsoid. The pitch value is applied in a counter-clockwise rotation over the item's pitch-axis. A pitch value of 45 makes the item face 45 degrees upwards, while a pitch of -20 makes it face 20 degrees downwards.
magnitude	double	In meter per seconds, ranging from [0, ∞) where 0 is standing still relative to the Earth's rotation.

9.5 Aggregated entity message

Using aggregated messages is defined in Section 5.7. Table 22 is a wrapper message containing a map with all entities that require an aggregated update. The values of this map would be the defined entity messages in Section 7, with all described properties to be optional allowing the message to only contain the relevant changed properties.

Table 22 — Aggregation message properties

Property	Type	Description
id	string	Unique case-insensitive identifier of the aggregation update.
map	map<Aggr>	Map containing key-value pairs, all with unique keys: *key* – unique case-insensitive identifier of the entity. *value* – entity message where all properties are optional.
timestamp	long	Optional UNIX Epoch time in milliseconds marking the time the aggregated update was performed.

Annex A
(informative)

List of all architectural guidelines

Table A.1 — Prerequisites of a common simulation space infrastructure

No.	Guideline
G.1	A common simulation space infrastructure should be based on well-developed and robust message broker software, where messages are exchanged between simulators and other tools.
G.2	Connecting to a common simulation space infrastructure should be made possible for all applications that are needed in the common simulation space, by means of configurable adapters that provide a documented application programming interface (API) for sending messages to and receiving messages from the specific common simulation space.
G.3	The common simulation space infrastructure should be modularized, having its core functionality of a message system expandable with additional modules for administrative functions, large data usage, data storage, application authentication and authorization, message encoding and encryption, as well as other features that might be needed for a specific common simulation space.

Table A.2 — Configuration of a common simulation space

No.	Guideline
G.4	A common simulation space may exclude architectural guidelines defined inside this document. Since the guidelines inside this CWA are complementary within one sub-section, when excluding architectural guidelines, it is recommended to exclude the complete set of guidelines defined per sub-section instead of individual guidelines.
G.5	A common simulation space may include configuration guidelines for setting up that specific common simulation space.
G.6	Configuration guidelines should not conflict with the architectural guidelines included in this CWA but should serve as a specification on those guidelines.
G.7	A common simulation space may include additional message formats, i.e. not defined as part of this CWA, required to be sent over that specific common simulation space.
G.8	New common simulation space message formats should not semantically conflict with messages already defined inside this document.
G.9	When configuring a common simulation space, the following should be defined: — used common simulation space infrastructure; — connected applications;

	— used architectural guidelines; — used configuration guidelines; — used messages, including the properties inside each message; — used pre-computed data or application-specific configurations; — interactions defining which messages are sent and received by which connected application; — common simulation space infrastructure channel or topic topology for facilitating the interactions; — rules for conflict handling that might arise during interactions or handling of messages.
G.10	Connected applications should implement a translation between their own domain model and the messages they should send or receive inside the specific common simulation space.

Table A.3 — Recognition of connected applications

No.	Guideline
G.11	All connected applications should send out a heartbeat message (see Section 6.1) to the common simulation space, at a constant time interval agreed-upon in the configuration guidelines, to indicate the application is responsive in the common simulation space.
G.12	Each connected application interested in another connected application's responsiveness may listen to the heartbeat messages of that connected application.

Table A.4 — Management of sessions

No.	Guideline
G.13	Session management should be handled by one session manager.
G.14	The session manager should inform the common simulation space of the current session information via the session management message (see Section 6.2).
G.15	Each connected application interested in session-related information should listen to the session management messages of the session manager.
G.16	Each session state change should be under supervision of a person or staff responsible for the session, making sure all connected applications are ready for the state change.

Table A.5 — Handling of time

No.	Guideline
G.17	Time management should be handled by a single time manager.
G.18	Further definitions of time inside the common simulation space should be in the configuration

	guidelines.
G.19	The time manager should inform the common simulation space of the current time information via the time management message (see Section 6.3) at a fixed time interval defined in the configuration guidelines.
G.20	The time manager should provide the current time and speed with each time management message.
G.21	Each connected application interested in time-related information based on intervals, should listen to the time management messages of the time manager.
G.22	Connected applications receiving a time management message should use the provided simulation time and speed for correcting their own application time mechanism to the provided time.
G.23	Whenever the time manager has changed its time state (e.g. the simulation is paused), it should inform the common simulation space of the current time information via the time management message (see Section 6.3).
G.24	Each connected application interested in time-related information based on time states, should listen to the time management messages of the time manager.

Table A.6 — Entities

No.	Guideline
G.25	An entity message should have a unique identifiable name (named id), to be used for referral by all connected applications.
G.26	An entity message may have an optional timestamp property, to represent the moment in time the update happened.
G.27	An entity message may have an optional type property, allowing the connected application sending the message to provide further information on what type of simulation entity it is sharing.
G.28	An entity message may have an optional tags property consisting of a list of unique string keys with accompanied string values, allowing the connected application sending the message to provide specific details that are relevant for the current common simulation space configuration.
G.29	Whenever a connected application has created a new simulation entity that needs to be shared or the application has performed an update on a shared entity, the connected application should send the corresponding entity message with all relevant information to the common simulation space.
G.30	Unless configuration guidelines define otherwise, whenever a shared entity is no longer to be shared on the common simulation space, that entity can become marked as deleted by means of an entity deletion message (see Table 16).

Table A.7 — Aggregation of entity data

No.	Guideline
G.31	Aggregation messages should only be used to provide updates to already shared entities.
G.32	Each defined entity message should also have its aggregated version containing a map with unique identifiers of the updated entities as a key, and entity aggregation information containing a subset of properties of the entity that can be changed throughout a session as a value.
G.33	All properties of entity aggregation messages should be marked as optional, to allow connected applications to send an update of a subset of these properties.

Table A.8 — Requests

No.	Guideline
G.34	A request message should have a unique identifiable name (named id), to be used for referral by all connected applications.
G.35	A request message should have an applicant property containing the identifier of the applicant.
G.36	A request message may have an optional tag property consisting of a list of unique string keys with accompanied string values, allowing the applicant to provide specific details that are relevant for the current common simulation space configuration.
G.37	Unless configuration guidelines define otherwise, any connected application may send out a request used in the common simulation space.
G.38	Unless configuration guidelines define otherwise, whenever a request is no longer applicable in a common simulation space, it may be cancelled by the applicant via a request cancellation message (see Table 17).
G.39	Each connected application that was handling the request and that received a request cancellation message concerning this request should, unless configuration guidelines define otherwise, stop current handling and leave the affected entities as is.

Table A.9 — Responding to requests

No.	Guideline
G.40	A response message should have a unique identifiable name (named id), to be used for referral by all connected applications.
G.41	A response message should have a request property containing the identifier of the request this message is responding to.
G.42	Configuration guidelines should clearly define further implications of the possible responses for all connected applications in the common simulation space.

Table A.10 — Transfer of ownership

No.	Guideline
G.43	An entity message should have an optional owner property (only when the entity can be shared) to represent the current owner of this simulation entity.
G.44	Whenever a connected application creates an entity that can have multiple owners (but only one at a time) inside a common simulation space, the owner property of that entity should be set to the identifier of the connected application, marking this connected application as owner.
G.45	Connected applications should only send (aggregated) entity messages for entities that they are owner of.
G.46	Whenever a connected application wants to own a simulation entity that it does not own, the connected application should send a request for ownership (see Section 8.1).
G.47	Unless configuration guidelines define otherwise, an owner receiving a request for ownership of a simulation entity it owns, should send out an entity message with its owner property changed to the applicant (thereby handing over the ownership to that applicant).

Annex B
(informative)

Use cases

B.1 Use case 1 – Full-scale and/or live exercise

This use case describes the use of the common simulation space to link up both simulators and several exercise-support tools. This is a large-scale outdoor exercise in which exercise staff (EXCON), role-players, participants and observers are dispersed over a large live-exercise terrain. In this example, the exercise has its goal to train the participants how to deal with a hacker attempting to shut down parts of the electricity grid combined with a public violent and chaotic event (e.g. riots or looting).

Basic flow

In this exercise, EXCON uses an exercise control tool (ECT) to control the exercise's timetable/playbook to steer the role-players and simulators from their enclosed control room. This tool also presents EXCON with the (summarized) observations from observers in the field. These observers make use of an observer tool (OT) to collect digitized human observations of the behavior of participants and how they deal with the scenario injects. This tool shares these observations with the exercise control tool, such that EXCON at their off-site location get a real-time overview of how the participants are managing the crisis in the field. Because evaluation of the actions of the participants is important for these large-scale exercises, all information that can be stored will be stored by a storage tool (ST). Furthermore, a simulator is used to model the electricity grid (simElec).

Figure B.1, Figure B.2, Figure B.3, and Figure B.4 depict a basic flow of the connected applications inside the common simulation space, including specific exercise tools, that all serve the purpose of facilitating a large-scale exercise being played out. These figures are unified modeling language sequence diagrams, displaying interactions between components (arrows between the columns) over time (progressing from top to bottom).

In Figure B.1, the start-up flow of the common simulation space during the exercise is further explained. Since EXCON is controlling the exercise, they are using the exercise control tool to inform all other parties that might be in other sections of the exercise field. Whenever initializing the exercise, the exercise control tool is used to inform all role players to get ready, since the exercise is about to start. This is also done for the observers via their own specific observer tool using the common simulation space session management message. The simulator will also receive the session management message with its state property set to value "Initialization" and initializes its simulation domain. The storage tool will receive the session management message and starts a new log for recording all common simulation space messages being sent.

Whenever EXCON decides to start the exercise, they first do a green light check to make sure all applications and persons are ready for the exercise. When everyone is ready, the exercise control tool is used for sending out the start of the exercise to all involved parties. Role-players are informed directly, and the session management message with its state property set to value "Start" is sent to the common simulation space. Although the electricity simulator receives this message, it will wait with starting the simulation until a time management message with the state "Started" has been sent. The usage of both the session and time management messages within this use case is done to make a clear distinction between the exercise session and the simulation itself. After the time management message has been

sent out as well by the exercise control tool, the electricity simulator starts its simulation and EXCON can inform the participants the exercise will start.

Due to planned breaks, or unforeseen issues, EXCON wants to have a pause during the exercise (depicted in Figure B.2), by first informing the participants, secondly by sending the pause command to the exercise control tool. This tool reports the pause to the role-players and sends out a time management message with the state property set to value "Paused". Received by the observer tool, the observers also receive the pause command. The simulator receives the time management message and pauses its running simulation. After EXCON receives the feedback the exercise is paused, they can perform the tasks required during this pause break. Based on the decision by EXCON the exercise will be resumed with a similar flow as the start of the simulation time.

Inside the running exercise a pre-defined internal script is triggered inside the electricity simulator, disabling all electric devices used by the exercise participants in grid A of the exercise (see Figure B.3). This is done via a direct connection between the electricity simulator and fuse boxes used for this exercise, resulting in the desired result. An additional flow is depicted to even further escalate the incident of this exercise. The observers report back that the participants are responding correctly to the situation at hand. This is done via the observer tool, sending out answer messages to be received by the exercise control tool. This allows EXCON to make the decision to further escalate the problem and put more pressure on the participants. Escalation is done via the exercise control tool, informing the role players of this decision and sending out a change questions request message for the observer tool to alter the current set of questions into a new set, more focused on the escalation aspects. The observer tool changes the questions for all the observers. Then a start inject request message is sent to inform the electricity simulator to start the specified escalation inject. Upon receiving this request, the electricity simulator disables all electric devices for all participants in grid B as well.

After some time, the participants have isolated the hacker's action and are able to contact electricity companies (acted out by the role-players) to restore the power. The role-players inform EXCON of this via the exercise control tool and EXCON directly changes the simulation of the electricity simulator, causing the simulator to enable all electric devices for both grid A and B. After changing the state of the electricity simulator, EXCON indicates that the incident has been mitigated. The exercise control tool sends this to the role-players to inform them and requests to change the set of questions to a mitigation questionnaire in the observer tool.

When stopping the exercise in Figure B.4, EXCON first informs the participants directly to make sure they are not introducing new actions for the exercise. Via the exercise control tool, the role-players are informed that they can do their own evaluation and cool down. The exercise control tool also sends out the time management message with its state property set to value "Stopped" to the common simulation space. The observer tool receives this message and reports this to the observers who can answer a final set of evaluation questions. The simulator will receive the time management message and stops its simulation. For the simulator, this would end its use for this exercise although the exercise isn't formally stopped and closed yet. EXCON is informed via the exercise control tool that the simulation time has been stopped and simulators are not running anymore. Observers can, however, still answer the evaluation questions to collect all data belonging to this exercise. Whenever EXCON knows all observers completed the evaluation, the complete exercise will be stopped and closed via the session management messages. This will let the exercise control tool send these final session management messages with its state property set to value "Stop" and "Close" respectively. The observer tool receives these messages and closes the observer tool session. The storage tool will save all recorded common simulation space messages into a persistent storage, assuring the data collected throughout this exercise can be used for evaluation purposes.

Architectural guidelines

In order to configure the common simulation space that can implement the above-mentioned use case, the following Sections of architectural guidelines are used:

— 5.1 – Prerequisites of the common simulation space infrastructure

— 5.2 – Configuration of the common simulation space

— 5.4 – Management of sessions

— 5.5 – Handling of time

— 5.8 – Requests

The following Sections are used for the described use case flow:

— 6.2 – Session management message

— 6.3 – Time management message

— 8.4 – Start inject request message

A common simulation space-specific answer message is introduced for this use case, containing three properties:

— the identifier of the observer that answered a question via the tool

— the question that is answered by the observer

— the answer of the observer belonging to the question

A common simulation space-specific change question request message is introduced for this use case, containing the following properties:

— the identifier of the request

— the applicant of the request

— the list of observer identifiers this change should apply to

— the identifier of the list of questions that the observer tool needs to change to

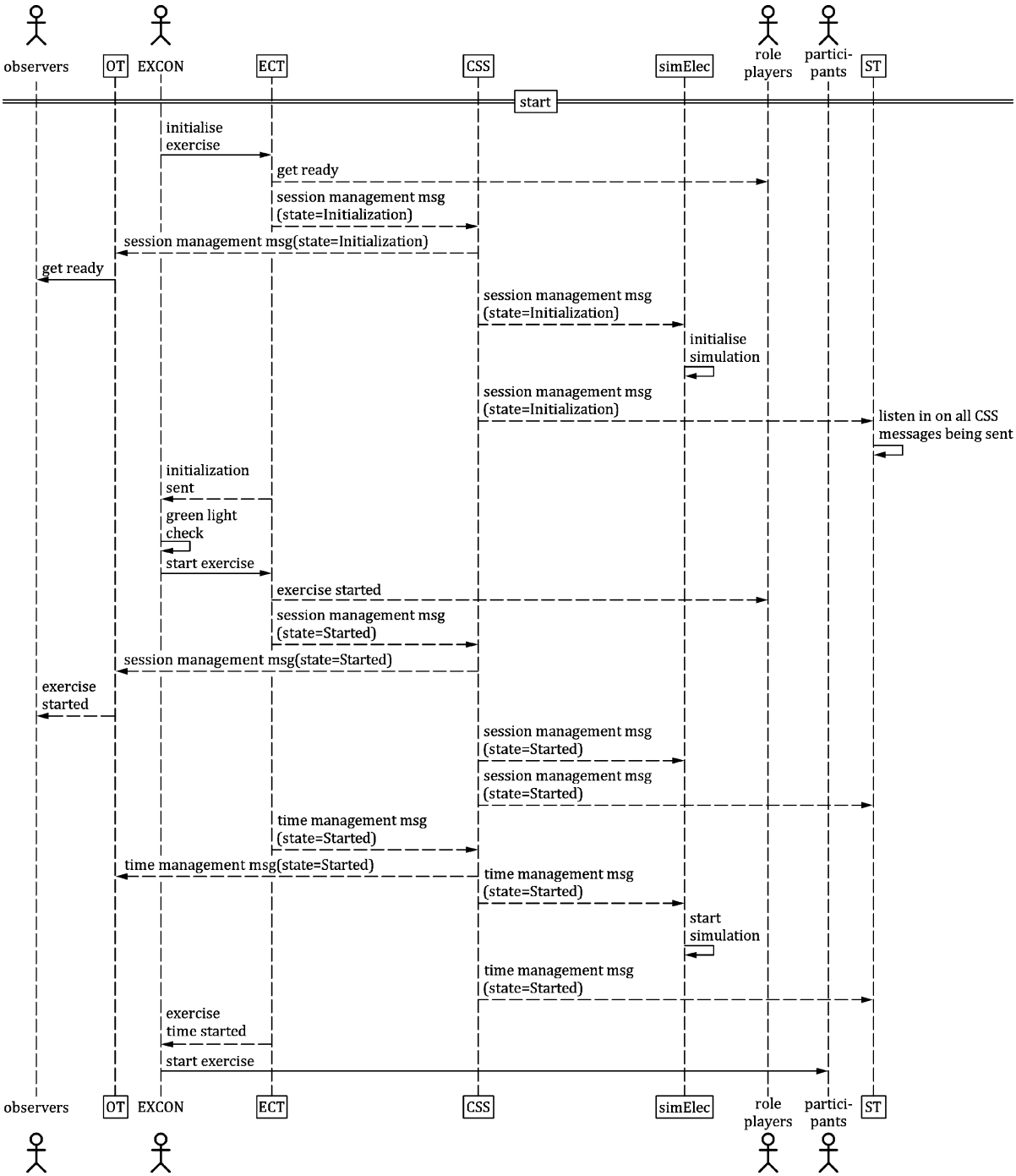

Figure B.1 — Initializing and starting the common simulation space inside an exercise

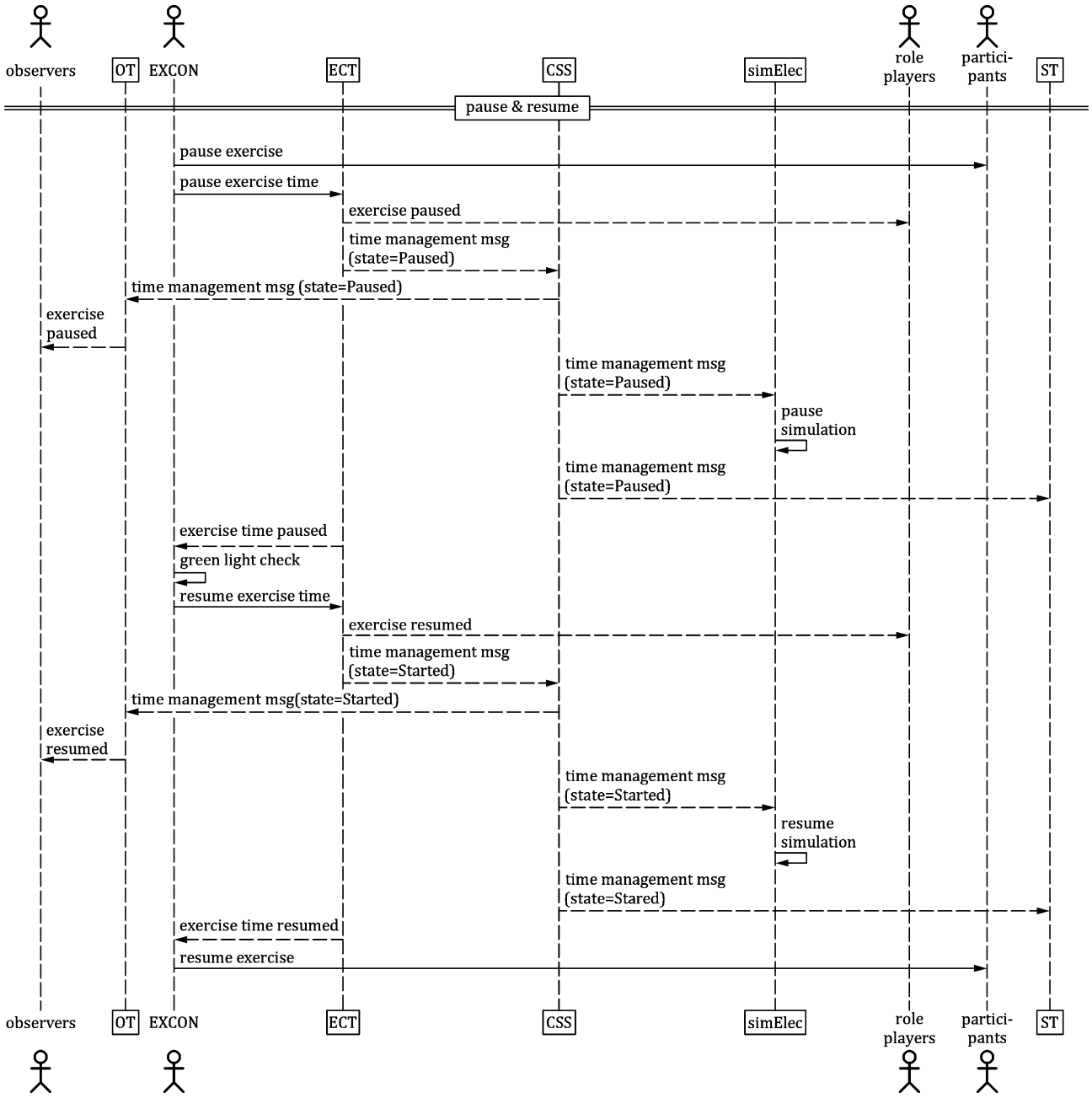

Figure B.2 — Pausing and resuming simulation time

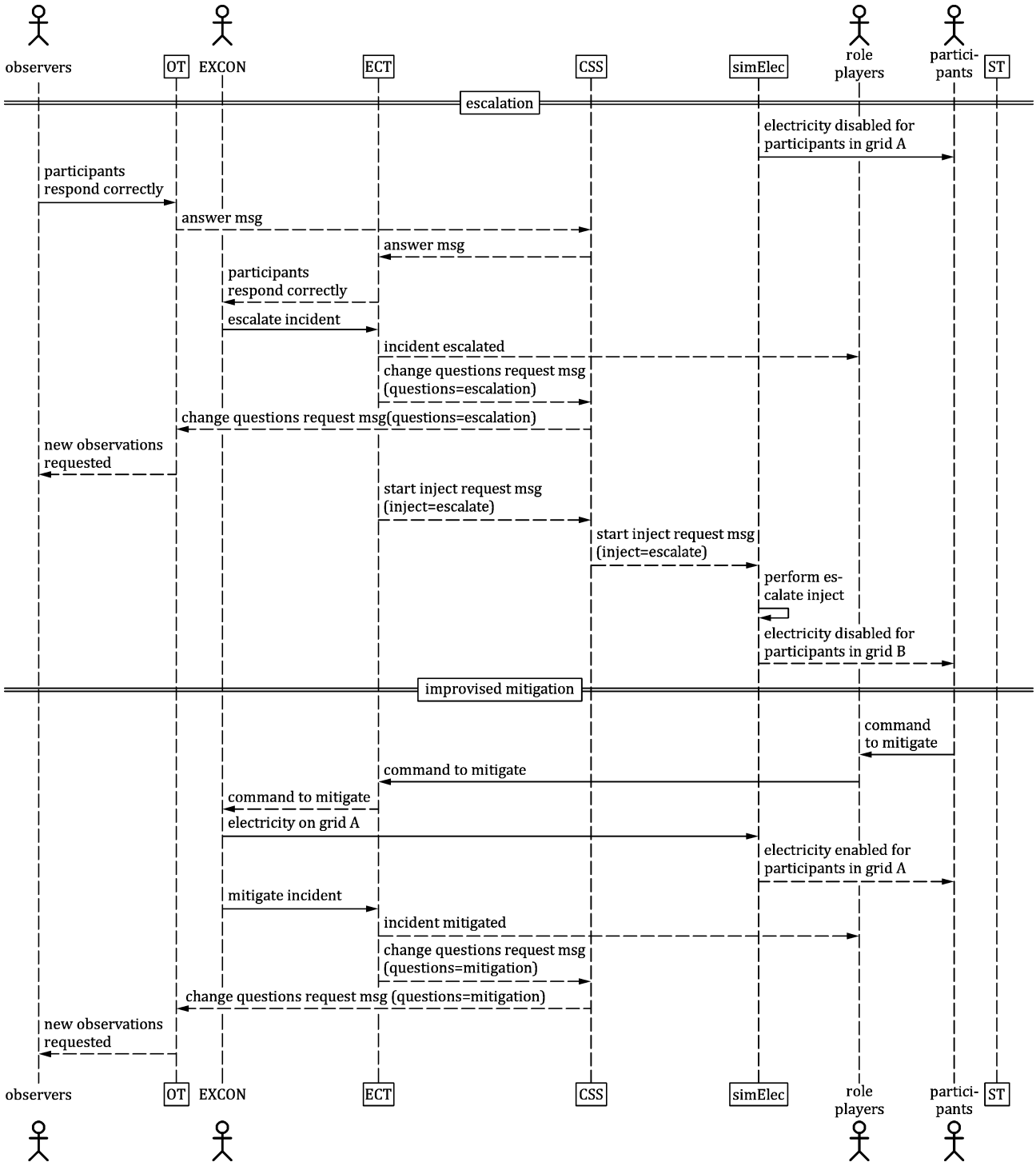

Figure B.3 — Simulator interaction and escalation of the exercise incident, including mitigation

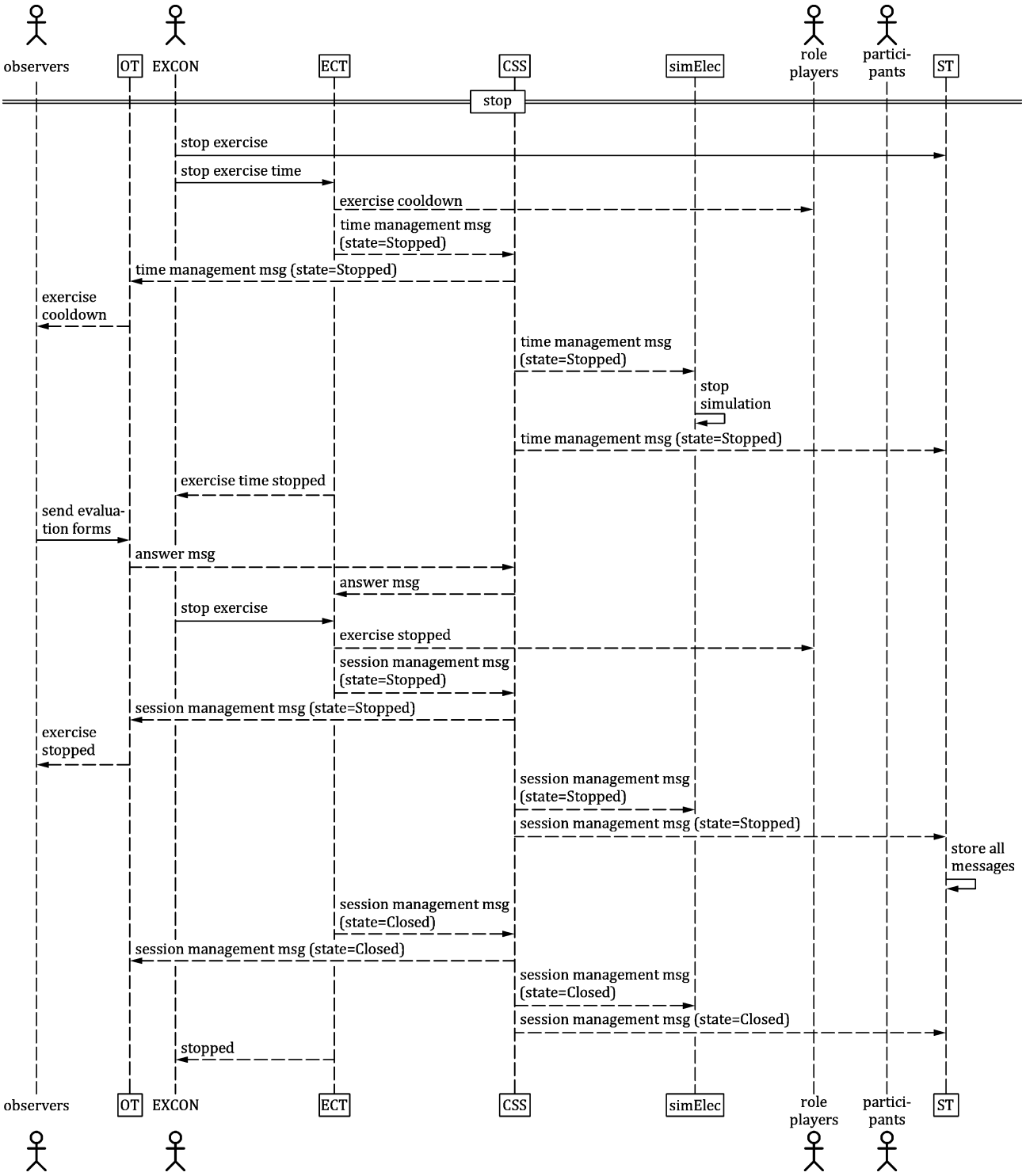

Figure B.4 — Closing down the exercise

B.2 Use case 2 – Cascading effects

This use case describes the processing of shared information between multiple simulators that influence these simulators. Individual pieces of information originating from a simulator serve as input for another simulator connected to the common simulation space, forming a cascading effect chain enriching the simulated world.

Basic flow

The scenario for the simulated world in this use case will be a flooding incident. Water rises because of a breach in an embankment (simulated by "simFlood" in Figure B.5), which causes cascading effects like power outages (simulated by "simElectricity" in Figure B.5), traffic congestion (simulated by "simTraffic" in Figure B.5), emergency response (simulated by "simResponder" in Figure B.5) and mitigating measures deployment (simulated by "simMitigation" in Figure B.5). The mitigating measures would in turn influence the water levels at certain affected areas, causing a circular cascading effect for all simulators involved.

Figure B.5 depicts a basic flow of the connected applications inside the common simulation space to update each other and react on those updates to facilitate the above-mentioned use case. This figure is a unified modeling language sequence diagram, displaying interactions between components (arrows between the columns) over time (progressing from top to bottom).

In Figure B.5, the simulation session and time has already started, and the flooding simulator is updating the flooded area for a particular time period. Whenever a new update is calculated, the flooding simulator will send out a new feature collection message containing all information about the flooded area to the common simulation space. Received by all other simulators, they will process this new data into their own simulation domain models. Based on the update from the flooding simulator, the electricity simulator can calculate how this new input affects the electricity grid. If any updates are found, the electricity simulator sends out a feature collection message describing areas of electricity outage to the common simulation space. Like the flood area, the other simulators can process this new information inside their own simulation domain. The same flow goes for the traffic simulator, which relies on both the flood area and the electricity outage areas to calculate possible traffic congestion.

Upon user commands, the responder simulator will move responder units to position them correctly in mitigating the breach causing the flood. Moving units will require plotting a route avoiding the flooded area and the traffic congestion and should consider electricity outages to find working power sources for applying the mitigating measures. Each position update of units (represented by item messages) will be sent from the responder simulator, via the common simulation space, towards the mitigation simulator. Whenever the responder simulator has put all units on the locations the user provided, the simulator can then request the mitigation simulator to start to resolve the breach. This is done via a start inject request, requesting to start the inject called "mitigate". The mitigation simulator upon reception of the request accepts the request and starts calculating the result of this configuration. When done calculating, the mitigation simulator sends out the results via a feature collection message to the common simulation space. This message is then received by the flooding simulator to update the simulation domain flooding model, possibly causing another chain of cascading effects as described above.

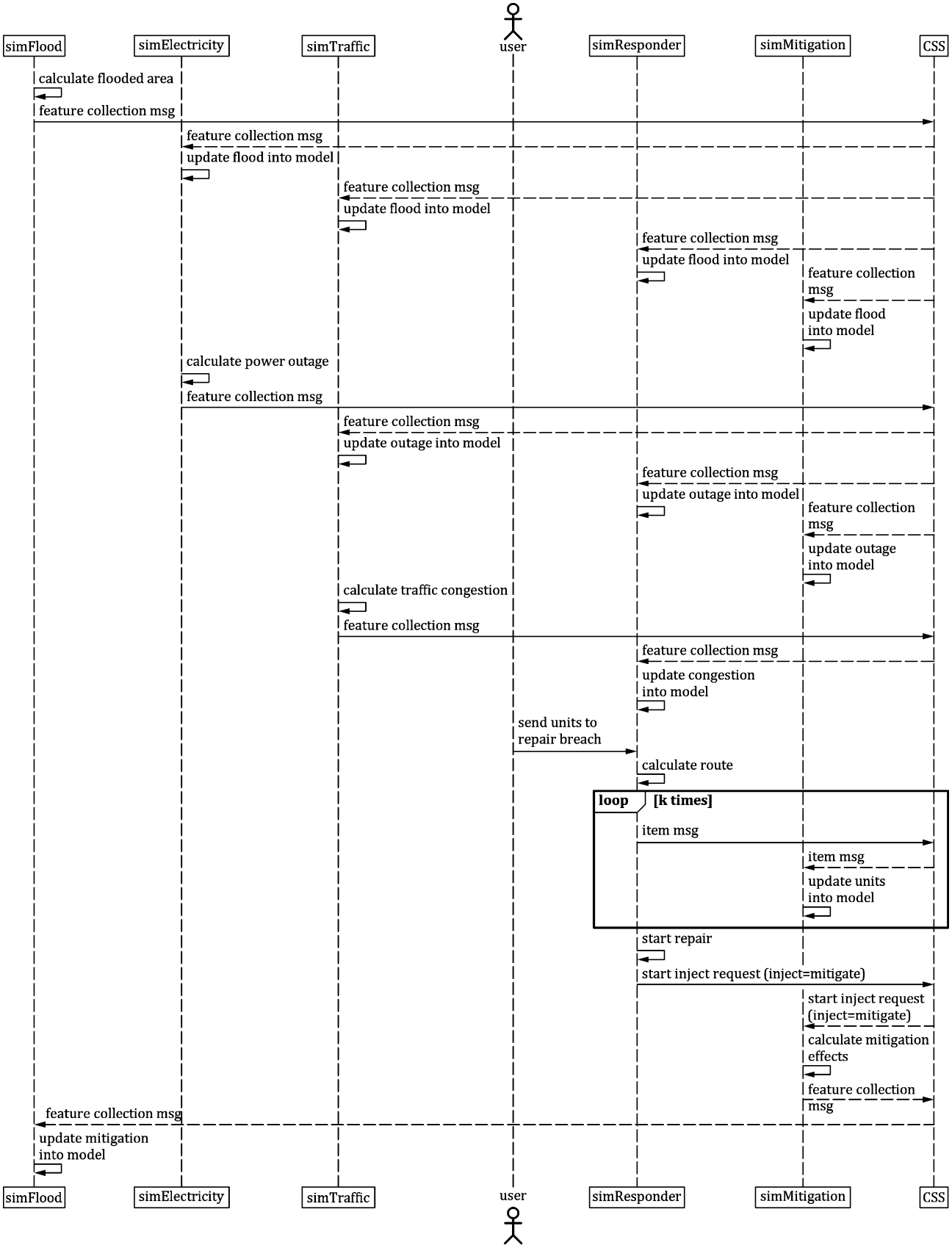

Figure B.5 — Cascading effect between multiple simulators reacting to each other's data updates

Architectural guidelines

In order to configure the common simulation space that can implement the above-mentioned use case, the following Sections of architectural guidelines are used:

— 5.1 – Prerequisites of the common simulation space infrastructure

— 5.2 – Configuration of the common simulation space

— 5.6 – Entities

— 5.8 – Requests

The following Sections are used for the described use case flow:

— 7.1 – Item message

— 7.2 – Feature collection message

— 8.4 – Start inject request message

B.3 Use case 3 – Ownership

This use case describes the interaction between two simulators that can transfer ownership of one of the shared entities. Because functionality is separated over multiple simulators, this use case describes a basic flow of how shared entities can change owner in order to perform a simulator-specific functionality.

Basic flow

The two simulators in this case are a visualization (depicted as "simVisual" in Figure B.6 and Figure B.7) and a routing simulator (depicted as "simRoute" in Figure B.6 and Figure B.7). The visualization simulator provides a user interface to visualize and move entities inside its simulation domain. Since the visualization simulator does not implement any movement of entities by itself, the routing simulator is used to do the actual transportation. An implementation inside the visualization simulator is created to enable the user to visualize the calculated transport route requested from the routing simulator and send out a transport request for transporting the item over the route.

Figure B.6 and Figure B.7 depict a basic flow of the connected applications in the common simulation space to transport an item and apply a minor change to the item being transported with use of several guidelines and messages defined in this document. These figures are unified modeling language sequence diagrams, displaying interactions between components (arrows between the columns) over time (progressing from top to bottom).

The simulation session and time has already started, and the visualization simulator is sending out updates regarding shared entities. In Figure B.6, the user selects an item to be transported inside the visualization simulator. Upon having entered where this item should be transported to, the visualization simulator sends out a route request to the routing simulator. This simulator calculates the route based on the request and sends a feature collection message containing a line depicting the suggested route.

Having seen the suggested route inside the visualization simulator, the user orders the transportation according to the suggested route. The visualization simulator sends out a move request message to the common simulation space. The routing simulator investigates if it can perform the transport request

from information of the item and destination obtained by previously sent entity messages. In order to perform the request, the routing simulator sends out an ownership request for the visualization simulator to hand off ownership. Whenever the routing simulator received this updated item message, it can start by updating the location of that item according to the calculated route towards the destination.

While the transport is still ongoing, the user has notified the visualization simulator to change the visibility of the item (see Figure B.7). Since this simulator does not own the item, it sends out an ownership request to the common simulation space. The routing simulator accepts this request, pauses the internal movement process and changes the owner property to the visualization simulator. The visualization simulator can now change the visibility, and immediately transfer back the ownership to the routing simulator, for it to continue the transportation.

After completing the move request, the routing simulator transfers ownership back to the visualization simulator, indicating it has completed the request. The visualization simulator can in turn report this back to the user of that simulator.

Architectural guidelines

In order to configure the common simulation space that can implement the above-mentioned use case, the following Sections of architectural guidelines are used:

— 5.1 – Prerequisites of the common simulation space infrastructure

— 5.2 – Configuration of the common simulation space

— 5.6 – Entities

— 5.8 – Requests

— 5.10 – Transfer of ownership

The following Sections are used for the described use case flow:

— 7.1 – Item message

— 7.2 – Feature collection message

— 8.1 – Ownership request message

— 8.2 – Route request message

— 8.3 – Move request message

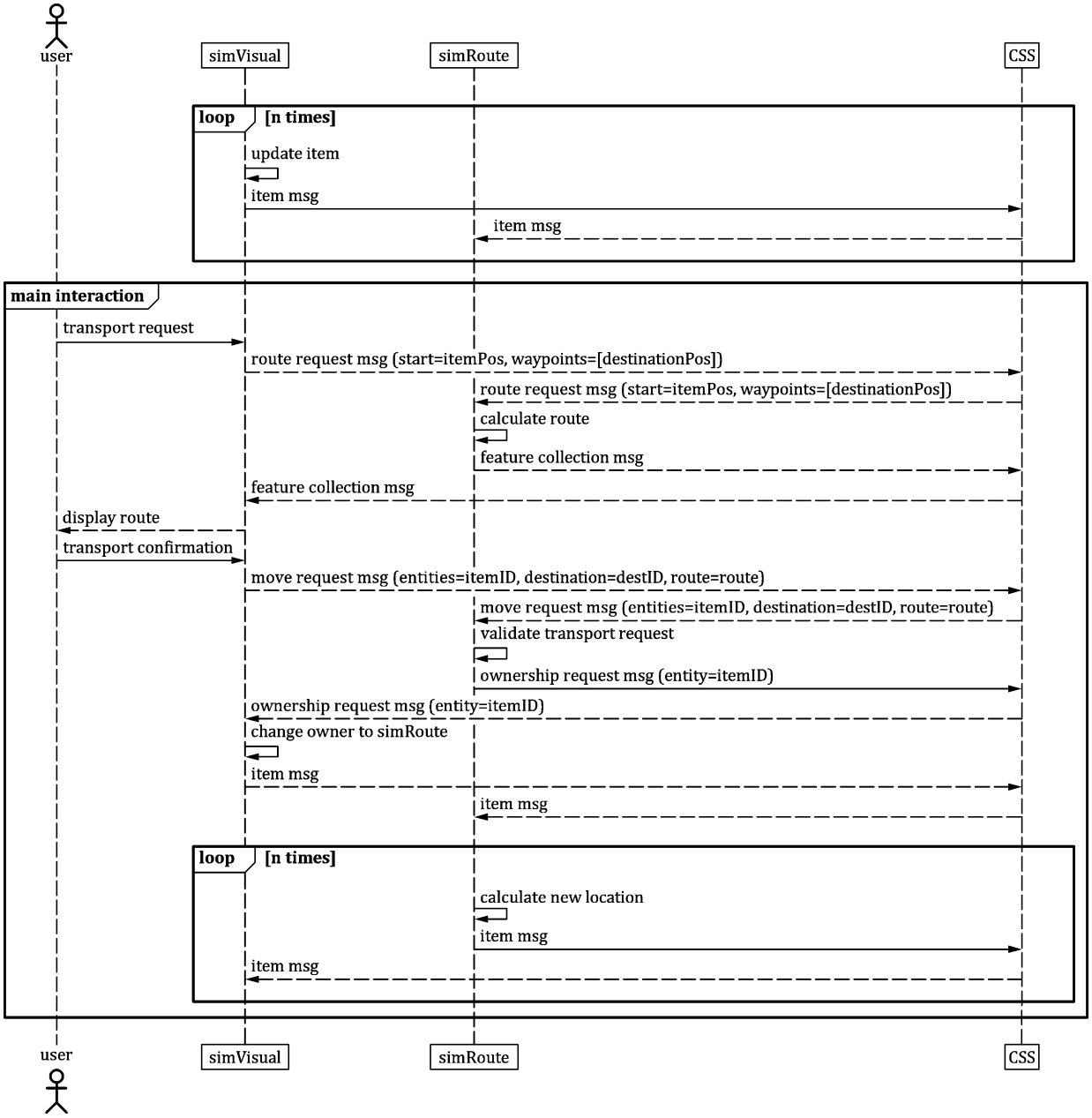

Figure B.6 — Requesting for transport of an item that requires a transfer of ownership

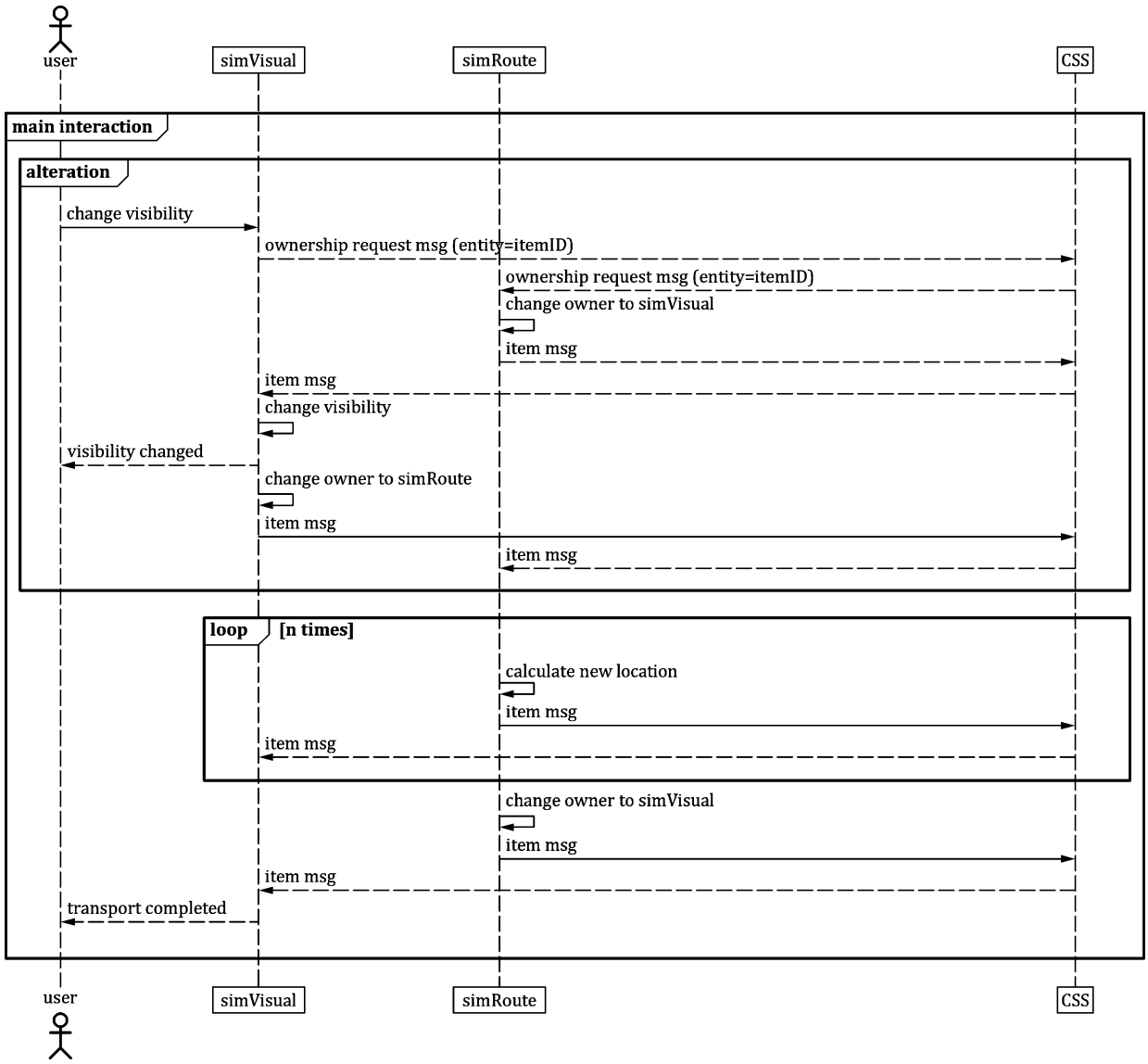

Figure B.7 — Small alteration within the transport

B.4 Use case 4 – High throughput

This use case describes the interoperability of two simulators that have many messages to transfer and process each time step. Both simulators have high fidelity and throughput requirements that need to be met in order to function properly in the common simulation space. To keep track of the synchronized time steps, a time manager is inside the common simulation space sending out time step information to allow time synchronization between both simulators.

Basic flow

The two simulators in this case are a crowd simulator (depicted as "simCrowd" in Figure B.8) and a traffic simulator (depicted as "simTraffic" in Figure B.8), both having many pedestrian and vehicle entities to simulate. All entities are inside a shared environment, and the simulators need to exchange information on current and near-future states to avoid any unwanted collisions between moving pedestrians and vehicles.

Figure B.8 depicts a basic flow of high throughput between the connected applications inside the common simulation space by use of guidelines and messages defined inside this document. This figure is a unified modeling language sequence diagram, displaying interactions between components (arrows between the columns) over time (progressing from top to bottom).

In this figure, the simulation is being started by the time manager by sending out a time management message to the common simulation space. The common simulation space propagates this message to all connected applications, and both simulators will start their first simulation time step. The crowd simulator sends out an aggregation message containing a map of all shared pedestrian items that are updated within this time step. The representation of a pedestrian by both simulators is agreed-upon as a circle with a fixed radius, so only the current location and velocity will be sent inside the aggregation message. The common simulation space propagates this to the traffic simulator, which processes the pedestrian data to be represented inside its own simulation domain. The traffic simulator in its turn sends out an aggregation message (via the common simulation space) to the crowd simulator, containing all information of shared and updated vehicle items. Representation of a vehicle for this common simulation space is defined as a rectangle, so the traffic simulator also sends the orientation changes of its vehicles to be processed by the crowd simulator. In the figure, a similar flow is provided for consecutive time steps.

Whenever the time manager wants to stop the simulation time, it sends out a time management message with its state property set to its value "Stopped". Propagated by the common simulation space and received by the simulators, they can both stop their own simulation.

In this basic flow, every simulator only provides updates of the given properties of the entities it owns, and the other simulator tries to incorporate them inside its own simulation domain. If the receiving simulator cannot integrate the information, it decides for itself how to resolve these issues. There is no explicit conflict resolution mechanism defined for this use case, although this can be provided by additional configuration guidelines.

Architectural guidelines

In order to configure the common simulation space that can implement the above-mentioned use case, the following Sections of architectural guidelines are used:

— 5.1 – Prerequisites of a common simulation space infrastructure

— 5.2 – Configuration of a common simulation space

— 5.5 – Handling of time

— 5.6 – Entities

— 5.7 – Aggregation of entity data

The following Sections are used for the described use case flow:

— 6.3 – Time management message

— 9.5 – Aggregated entity message

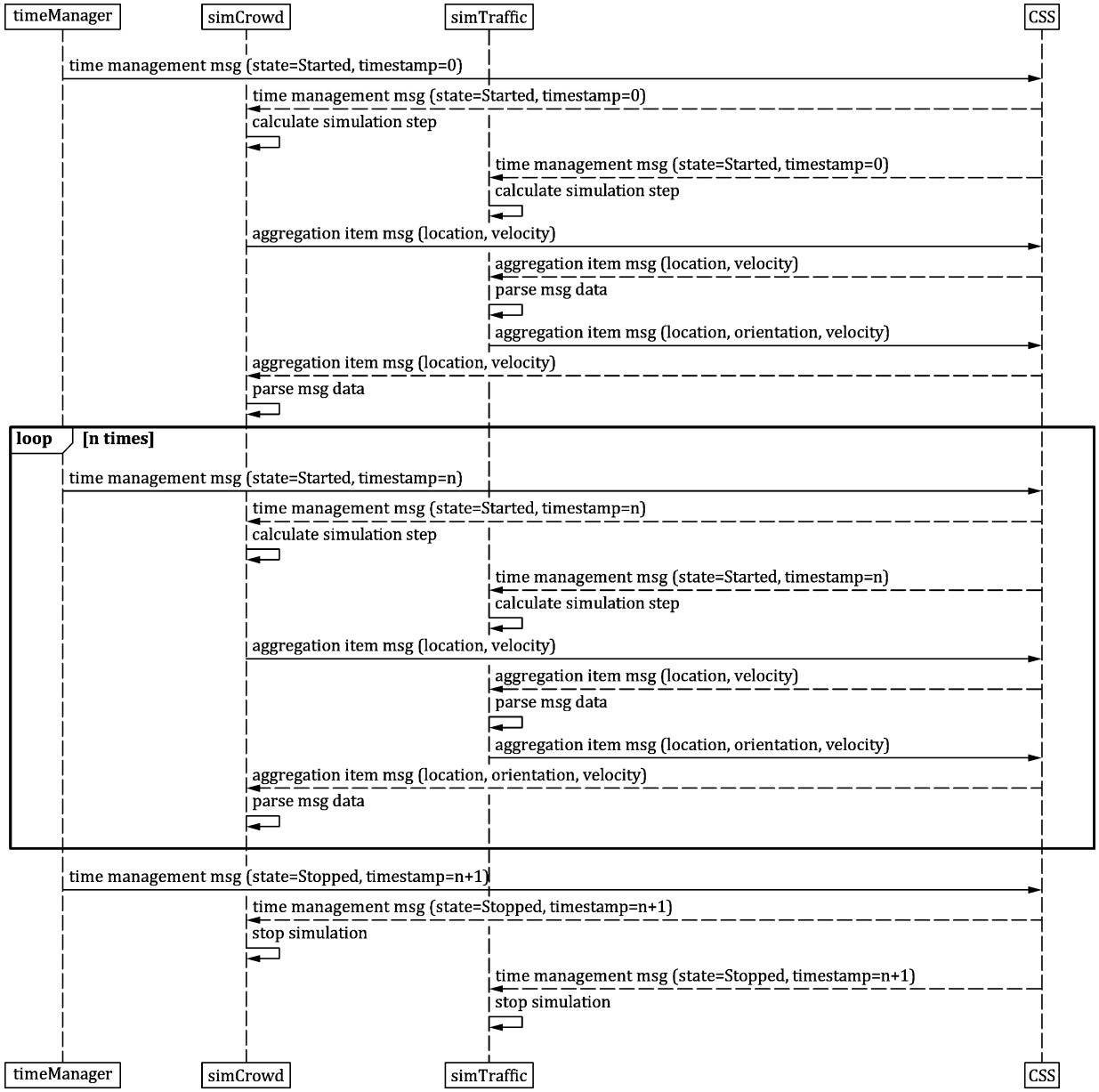

Figure B.8 — High throughput between two simulators exchanging aggregation messages

B.5 Use case 5 – Interaction with an external simulation space

This use case describes the interaction between connected applications inside the common simulation space, with one or more simulators connected via an external simulation space (ESS) (e.g. a high-level architecture implementation). The focus of this use case regards guidelines for a gateway that needs to be created between the two simulation spaces. It also introduces the use of the response message to provide an example of handling session initialization in a more automated form (i.e. excluding a person responsible for checking if all connected applications are ready).

Basic flow

In this basic flow, simulator A (depicted as "simA" in Figure B.9) and the application serving as the gateway between the common simulation space and the external simulation space (depicted as "CSS-ESS gateway" in Figure B.9) are both connected to the common simulation space. The gateway application is also connected to the external simulation space, where simulator B (depicted as "simB" in

Figure B.9) resides. An exercise control tool (depicted as "ECT" in Figure B.9) provides the external simulation space with a simple session management that should also control the common simulation space session management.

Figure B.9 describes a basic flow of simple simulator interactions between the common simulation space and the external simulation space via the gateway application connected to both spaces. This figure is a unified modeling language sequence diagram, displaying interactions between components (arrows between the columns) over time (progressing from top to bottom).

In this figure, the exercise control tool – serving as session manager of the external simulation space – sends out a command for starting the simulation, automatically starting simulator B upon receiving this message. In this specific use case, simulator A requires some time to initialize its simulation domain and the gateway application needs to serve as a session manager inside the common simulation space. It first sends out the session management message with its state property set to value "Initialization" and waits until the simulator has responded positively. This allows simulator A to load and configure its simulation domain before sending out a response to the session management message. If any update messages are to be sent by the external simulation space, the gateway application should cache these messages until simulator A has started its simulation. Whenever properly initialized, simulator A responds to the session management message with an appropriate response message, for the gateway application to send out the session management message with its state property set to value "Start", possibly followed by the cached messages. After this, both external simulation space and common simulation space are in synchronization again and messages can be exchanged between the two spaces.

From common simulation space to external simulation space, all relevant entity and request messages from simulator A need to be translated by the gateway application to be received by simulator B. In order to translate a transport request into a proper transport command for simulator B, the gateway application needs to translate the references to the individual components inside the request (i.e. the simulation entity to be transported and the simulation entity that serves as the destination). Whenever this translation is performed correctly, the gateway application can send out the transport command to simulator B. From external simulation space to common simulation space, the same translation procedures apply for the gateway application.

Most likely, each external simulation space has a different set of guidelines and messages used to operate. Via a gateway application like the one proposed inside this use case, a bridge can be created between an external simulation space and a common simulation space, although this might require a large effort for simulators to set up correct connecting configuration guidelines, and for developing the gateway application based on these configuration guidelines.

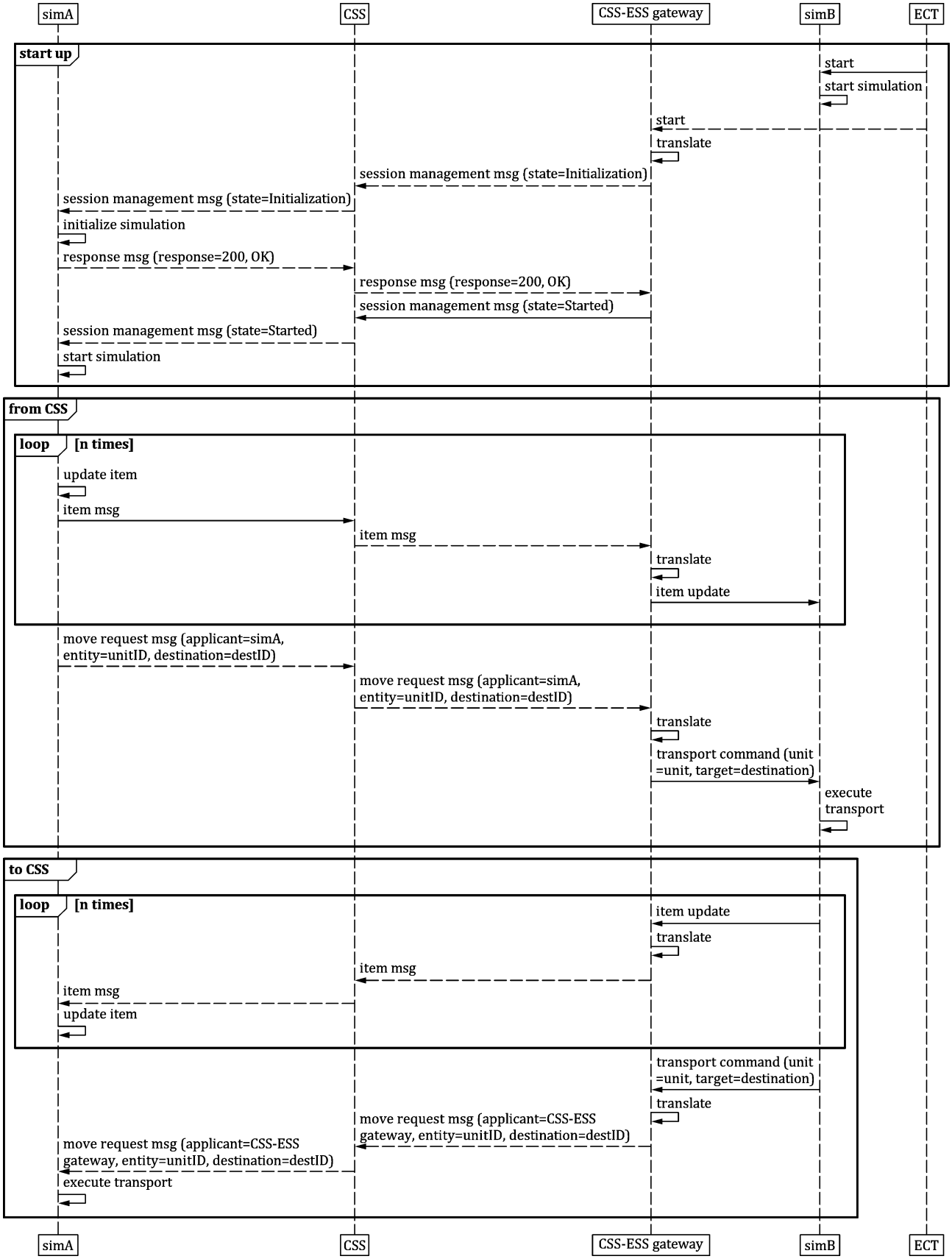

Figure B.9 — Communication between simulator A inside a common simulation space and simulator B inside an external simulation space

Architectural guidelines

In order to configure the common simulation space that can implement the above-mentioned use case, the following Sections of architectural guidelines are used:

— 5.1 – Prerequisites of the common simulation space infrastructure

— 5.2 – Configuration of the common simulation space

— 5.4 – Management of sessions

— 5.6 – Entities

— 5.8 – Requests

— 5.9 – Responding to requests

The following Section are used for the described use case flow:

— 6.2 – Session management message

— 7.1 – Item message

— 8.3 – Move request message

— 9.3 – Response message

B.6 Use case 6 – Interaction between common simulation and a shared application space

This use case describes the interaction between connected applications inside the common simulation space, with one or more user applications connected via a shared application space (SAS). The focus of this use case regards guidelines for a gateway that needs to be created between the common simulation space and the standardized shared application space. For more information on the interoperability between different crisis management organizations and applications that can help in defining a shared application space, the reader is referred to CWA 17513:2020 [6].

Basic flow

In this use case, a flood monitoring application (depicted as "appFloodMonitor" in Figure B.10) is used by a participant. A flooding simulator (depicted by "simFlood" in Figure B.10) generates simulated water levels in the specified area of interest, in place of actual water sensor readings. The flood area calculated is provided to the common simulation space, received by a gateway application (depicted as "CSS-SAS gateway" in Figure B.10). The data of the message is stored in a shared application data component part of the gateway application, which can be accessed by all applications in the shared application space. Information about this new flood data is sent out via standard operational message formats used throughout the shared application space, like Common Alerting Protocol [4], or Emergency Services Messaging Interface (ESMI) [5].

Figure B.10 describes a basic flow of this flooding simulator inside the common simulation space that communicates with the gateway application to provide standardized sensor data to be visualized by the flood monitoring application inside the shared application space. This figure is a unified modeling language sequence diagram, displaying interactions between components (arrows between the columns) over time (progressing from top to bottom).

The session and possible time management have already started, and all applications are connected to the required spaces. This basic flow starts with the flooding simulator having an update of the flooded area, sending this via a feature collection message to the common simulation space. The gateway application receives this message and processes the information to be stored as water level measurements from sensors at the pre-defined locations that would normally be accessible at an operational setting. The gateway application also sends out a common alerting protocol (CAP) message to the shared application space to notify all applications inside this space via an operational standardized message format. The flood monitor application receives this message and displays the notification new flood information can be obtained. The user of the flood monitor application actively requests the latest sensor data by pressing the notification, which leads to a direct communication between flood monitor application and gateway application to retrieve the current flooding sensor data. Whenever the data is retrieved from the gateway application, the flood monitor application displays the new information to the user.

Interaction from shared application space to common simulation space is also very likely in this use case, since there might be a very large amount of flooding sensors in a region. In order to keep a proper overview, information is aggregated in both the flood monitor application and the flooding simulator. The flood monitor application has the functionality to focus on a specific area of interest that should then receive more detailed sensor information regarding that selected area. Since this is not a functionality that can be written in current standardized operational message formats, the flood monitor application communicates directly with the gateway application to provide the region of interest. Upon receiving this information, the gateway application converts this into a feature collection message with a special "focus" type to be send via to the common simulation space. The flooding simulator receives this area and calculates additional information regarding the area of interest and provides an update via a new feature collection message. The further flow regarding processing the new flooded area message is handled according to the description above.

Architectural guidelines

In order to configure the common simulation space that can implement the above-mentioned use case, the following Sections of architectural guidelines are used:

— 5.1 – Prerequisites of the common simulation space infrastructure

— 5.2 – Configuration of the common simulation space

— 5.6 – Entities

The following Sections are used for the described use case flow:

— 7.2 – Feature collection message

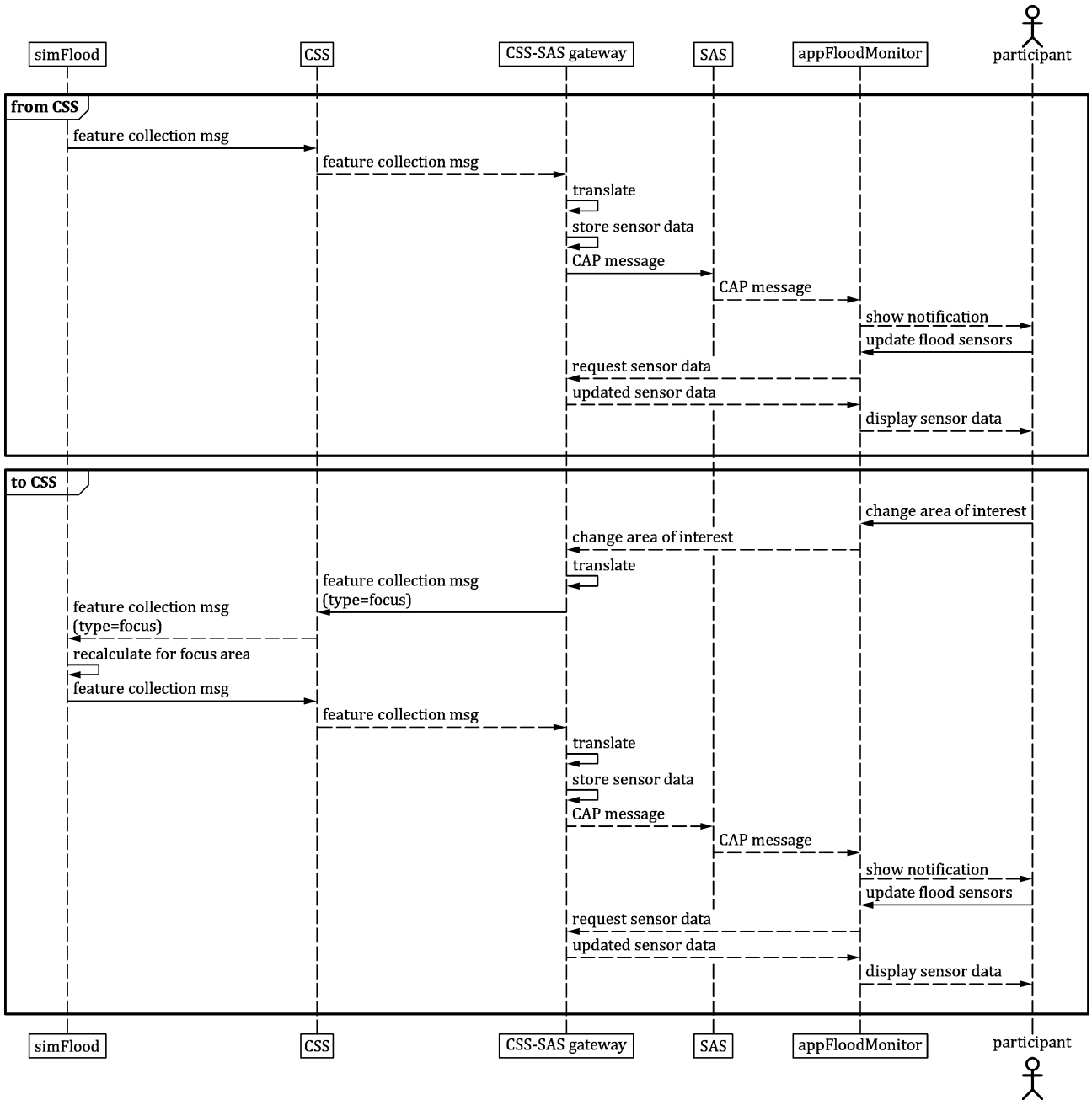

Figure B.10 — Communication between a flooding simulator in the common simulation space and a flood monitor application in the shared application space

Bibliography

[1] DRIVER+ website. Available at https://www.driver-project.eu/ [viewed 2020-03-16].

[2] Driver+ GitHub code repository. Available at https://github.com/driver-eu [viewed 2020-03-16].

[3] World Geodetic System 1984 (WGS 84). National Geospatial-Intelligence Agency (USA).

[4] Common Alerting Protocol (CAP). Available at http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html [viewed 2020-03-16].

[5] Emergency Services Messaging Interface (ESMI). Available at https://pos.driver-project.eu/en/standards/683 [viewed 2020-03-16].

[6] CWA 17513:2020 *Crisis and disaster management – Semantic and syntactical interoperability*.

[7] MapBox simplestyle-spec GitHub code repository. Available at https://github.com/mapbox/simplestyle-spec/tree/master/1.1.0 [viewed 2020-03-16].