Project number: 101111888
Project acronym: SHIMMER
Project duration: 09/2023 – 08/2026
Project Coordinator:
Andrea Alvarado Shmueli, SINTEF AS – Institute SINTEF Industry

Co-funded by the European Union

Clean Hydrogen Partnership

Horizon Europe Framework Programme (HORIZON)

HORIZON-JTI-CLEANH2-2022-05-03

Safe hydrogen injection management at network-wide level: towards European gas sector transition

Shimmer

Safe Hydrogen Injection Modelling and Management for European gas network Resilience

D4.4 Open-source fluid-dynamic model with gas quality tracking (handbook)

VERSION	DATE	TYPE	DISSEMINATION LEVEL
1.1	09.12.2025	Report	PU

ABSTRACT

One of the main objectives of this project is to develop an open-source fluid-dynamic model with gas quality tracking. Its name is Shimmer ++ and this deliverable serves as a handbook for installation and use, documenting model formulation, database schema, and numerical methods to support reproducible hydrogen-blending and renewable-gas integration studies, while providing the basis for future functional extensions.

Shimmer++ made available on a public GitHub repository. The model is implemented as an efficient C++ solver and structured to performs both steady-state and transient simulations and explicitly handles multi-species transport and admixing at network nodes. Gas thermophysical properties are computed using the GERG-2008 equation of state, enabling accurate evaluation of compressibility and supporting mixtures with up to 21 chemical species.

Designed to complement rather than replace commercial simulators, Shimmer++ emphasizes transparency and user extensibility. It follows a file-based approach without a graphical user interface, but the public GitHub repository provides examples for scripting and integration with external environments such as GIS From a user standpoint, the software adopts a three-layer architecture centred on Network Data Files (NDFs), i.e., SQLite databases storing topology, asset characteristics, boundary conditions, gas properties, and simulation outputs.

A typical workflow involves initializing an NDF, populating it with network data, specifying simulation settings via a lightweight Lua configuration file, running the solver, and post-processing results directly from the updated NDF.

AUTHORSHIP AND APPROVAL INFORMATION

AUTHOR(S)	DATE / SIGN
Marco Cavana / PoliTo Karol Cascavita / PoliTo Matteo Cicuttin / PoliTo Fabio Vicini / PoliTo Angelo Spadavecchia / PoliTo Luisa Di Francesco / PoliTo Stefano Berrone / PoliTo Pierluigi Leone / PoliTo	05.12.2025
REVIEWED BY WP-LEADER	**DATE / SIGN**
Huib Blokland / TNO	09.12.2025 ValidSigned by Huib Blokland on 2025-12-09 08:56:52
APPROVED BY COORDINATOR	**DATE / SIGN**
Diana González / SINTEF	09.12.2025 Diana González (Dec 9, 2025 10:07:32 GMT+1)

CO-FUNDED BY THE EUROPEAN UNION. VIEWS AND OPINIONS EXPRESSED ARE HOWEVER THOSE OF THE AUTHOR(S) ONLY AND DO NOT NECESSARILY REFLECT THOSE OF THE EUROPEAN UNION OR THE CLEAN HYDROGEN JOINT UNDERTAKING. NEITHER THE EUROPEAN UNION NOR THE GRANTING AUTHORITY CAN BE HELD RESPONSIBLE FOR THEM.

Release history

VERSION	DATE	VERSION DESCRIPTION
01	06.11.2025	First draft for review (in the form of Handbook)
1.0	05.12.2025	Final version
1.1	09.12.2025	Final version (typos corrections)

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

Table of Contents

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

Executive Summary

This deliverable reports the completion and release of the open-source fluid-dynamic model with gas quality tracking for the simulation of natural gas infrastructures under hydrogen blending scenarios. This is one of the main outcomes of the SHIMMER project, and was planned to support the ongoing transition of natural-gas infrastructures to renewable and low-carbon gases, especially hydrogen where gas quality tracking becomes essential for operational feasibility, safety and energy billing, requiring a detailed network-wide assessment. The model is called Shimmer++, and it is publicly accessible at the repository at `https://github.com/shimmerhydrogen/shimmer`. This includes the Shimmer++ sources and some documents related to the underlying physical model, along with the GitHub issues that record the developments of Shimmer++.

Shimmer++ is an efficient C++ computational tool for steady-state and transient (unsteady) simulation of transmission and distribution gas networks, including multi-species mixtures tracking and admixing at nodes. Up to 21 chemical species can be included as the model uses the GERG-2008 equation of state for the calculation of the gas compressibility factor. Shimmer++ is designed to enable quick, transparent simulation for distributed injections of non-conventional gases, complementing —rather than replacing— commercial software packages. The way it has been structured was intended to be as flexible and as customizable as possible, giving the users the possibility to personalize and expand its functions. The tool is file-based and does not include a graphical user interface. However, examples are given within the repository on how to set up scripts to facilitate integration with other environments, such as GIS or any other data visualization means. From a user perspective, Shimmer++ adopts a three-layer architecture centered on Network Data Files (NDFs): SQLite databases that store network topology, asset features, boundary conditions, gas properties, and simulation outputs. A typical workflow is: create/initialize an NDF, populate it with network data, define simulation parameters in a small Lua configuration file, run the solver, and post-process results directly from the updated NDF.

This document is intended to be the handbook to guide any future user through the use of Shimmer++. It documents the instructions for the installation, some theoretical background on model formulation, database schema, and numerical methods, providing researchers and practitioners with a reproducible reference to apply Shimmer++ to hydrogen blending and renewable-gas integration studies and even the basic information to optionally extend some functionalities.

About the project: The European natural gas infrastructure provides the opportunity to accept hydrogen (H_2), as a measure to integrate low-carbon gases while leveraging the existing gas network and contributing to decarbonisation. However, there are technical and regulatory gaps that should be closed, adaptations and investments to be made to ensure that multi-gas networks across Europe will be able to operate in a reliable and safe way while providing a highly controllable gas quality and required energy demand. Aspects such as material integrity of pipelines and components, as well as the lack of harmonisation of gas quality requirements at European level must be addressed in order to facilitate the injection of H_2 in the natural gas network.

In this context, the SHIMMER project (Safe Hydrogen Injection Modelling and Management for European gas network Resilience) was selected for funding as part of the 2023 Clean Hydrogen Partnership programme. SHIMMER aims to enable a higher integration of low-carbon gases and safer H_2 injection management in multi-gas networks by strengthening the knowledge base and improving the understanding of risks and opportunities in H_2 projects.

It will do this by:

- Mapping and assessing European gas T&D infrastructure in relation to materials, components, technology, and their readiness for hydrogen blends.
- Defining methods, tools and technologies for multi-gas network management and quality tracking, including simulation, prediction, and safe management of network operation in view of widespread hydrogen injection in a European-wide context.
- Proposing best practice guidelines for handling the safety of hydrogen in the natural gas infrastructure and managing the risks.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

1 Introduction

1.1 Purpose of the document

This report provides a comprehensive overview of the open-source fluid-dynamic model with quality tracking developed as a Shimmer project deliverable (D4.4). The open-source model's name is Simmer++. Being Shimmer++ itself the deliverable (deliverable type: "DATA") according to the project DoA, this document aims to:

- specify the open-source model location: it is publicly accessible at the GitHub repository: `https://github.com/shimmerhydrogen/shimmer`;
- give the detailed instructions on how to install and run Shimmer++;
- describe the Network Data Files (database schema) and how to create/populate them;
- illustrate the overall gas network modelling approach and indicate further scientific literature references for the detailed description;
- provide and in-depth description of the most relevant sections of the code, to allow advanced user to customize the tool.

1.2 Intended readership

The intended audience and the respective added benefit for each body includes:

- **Universities/training entities:** the model can be widely used as an educational tool without any restrictions and expenditures concerning license, fostering the knowledge on natural gas infrastructure, its transition and how to manage it.
- **Universities/Research Institutes:** the model is a ready-to-use tool for studies regarding the infrastructural transition towards hydrogen and renewable gases (biomethane, sNG...) without the commitment of buying a commercial software license. Unlike the commercial softwares, being open-source and publicly accesible allows for customization, improvement and integration with other other open-source models and tools (provided that the reserchers have a good background on gas network modelling and coding languages)..
- **System Operators (transmission and/or distribution)**: the model can be used within the research and development departments, preferably by persons who received preliminary training, to carry out strategic industrial research in a smart, flexible and agile way. Being open source, the product does not commit the company with expenses, does not constraint the working group to interaction and assistance with the software house and in general, gives a much more flexible and customizable tool (provided that the employees have a good background on gas network modelling and coding languages).

Readers of this document are expected to have existing knowledge about the transmission and distribution gas networks. Basic knowledge and skills about coding and open-source environment would ease the reader into the fast execution of some of the installation commands and the understanding of some coding parts. However, following the instructions given within this document, any common software user is enabled to use the model and the knowledge of C++ language is not necessary to run the model and get results from the desired simulation of case studies.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

1.3 Structure of this document

This document is organized as follows.

In Section 2, an overview of the background, the aims and the motivations behind the creation of an open-source simulation tool of gas networks is given. In Section 3, a thorough step by step installation procedure for Shimmer++ is shown. The description of the overall network model with the modeling of relevant network element is given in Section 4. Section 5 is devoted to the description of the database. The discussion about the numerical methods stage, including the model and the discrete setting for the mixing and transport of gas species are illustrated in Section 6. Finally, short conclusions and future exploitation possibilities are briefly commented in Section 7.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

2 Background, aims and motivation

In the framework of transitioning the natural gas infrastructure towards the inclusion of renewable gases such as biomethane and, overall, hydrogen, the need for simulation tools able to perform quality tracking at all network levels is becoming more and more urgent. For example, already in the case of biomethane, its injection on a local distribution network can cause a non-negligible perturbation of the gas quality. Even though in first approximation biomethane and natural gas can be treated as the same gas, when it comes to energy billing, the capability to distinguish two gases with different composition (and so different calorific values) can become a necessity, even at local level. This is even more true when hydrogen blending is considered as a possible integration pathway. Hydrogen, in fact, has completely different features than natural gas and it is not a natural compound of natural gas. Thus, even lower concentrations of hydrogen ($\sim 2\%$) can be unacceptable by some network users (e.g. gas refuelling stations).

The project SHIMMER - Safe Hydrogen Injection Modelling and Management for European gas network Resilience - has among its aims, the release of an open-source simulator of natural gas infrastructure with quality-tracking and admixing features, that can be use to simulate cases with distributed injection of renewable gases within a network infrastructure transporting or distributing natural gas with its specific composition. According to the ends of the project, the main focus is the simulation of hydrogen blending cases. The release of an open source tools with such capabilities would ease the academia, research companies and industries to handle scenarios of non-conventional use of their infrastructure, allowing them to address feasibility studies and, in general, acquire knowledge and experience in an open and quick way. On the other hand, this tool is not to be intended as a substitute of commercial softwares, which offer not only reliable tools but a variety of customization options and aftersale services which are out of the scope of this SHIMMER project outcome.

The aim of Shimmer++ development is to build an efficient open-source tool in C++ for the numerical modelling of natural gas transport through a long-distance network or natural gas distribution through local distribution networks. Shimmer++ is capable of steady and unsteady state simulations with mixture of gases, to take into account - for instance - hydrogen blending. Other tools exists, either open-source and commercial. As for open-source ones, to mention some: GasNetSim, pandapipes and MORGEN. The first was presented recently in `https://ieeexplore.ieee.org/document/9769148`, where authors argued GasNetSim is the first open-source tool that allows complex mixture composition of gases. This salient advantage can be only exploited for steady regime. The second library has less model flexibility and, as in the former, is written Python. The latter stands for Model Reduction for Gas and Energy Networks, which is an academic tool developed in Matlab based on the isothermal Euler equations and does not perform any quality tracking. As for commercial tools, the quality tracking feature is commonly implemented and used by Transmission System Operators (TSOs) in order to keep track of natural gas composition perturbations and/or perturbation in the calorific feature of the gas. Some of the most popular ones are SIMONE, OLGA and SAINT. To be noticed that these softwares are mainly used by TSOs on their regional or nation wide networks, while local Distribution System Operators (DSOs) usually do not need such advanced features as the natural gas flowing in their infrastructure is commonly homogeneous in composition.

Shimmer++ is a purely computational tool mainly intended to interact with other software, for example GIS programs. Therefore, Shimmer++ does not provide a graphical user interface and the interaction is file-based.

The Shimmer++ tool is based on the architecture shown in Fig. 1. The main functional domains are the on-disk gas network representation, the in-memory representation, and the numerical methods operating on the networks. From the users' point of view this three-layer architecture translates to the following:

- the on-disk representation is realized by the the Network Data Files (NDFs), which are SQLite databases where the gas networks are stored and are the main interaction mechanism between Shimmer++ and the user (where the user is potentially another software)
- the in-memory representation and the numerical methods layer are the actual shimmer++ tool, which reads and writes NDFs

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

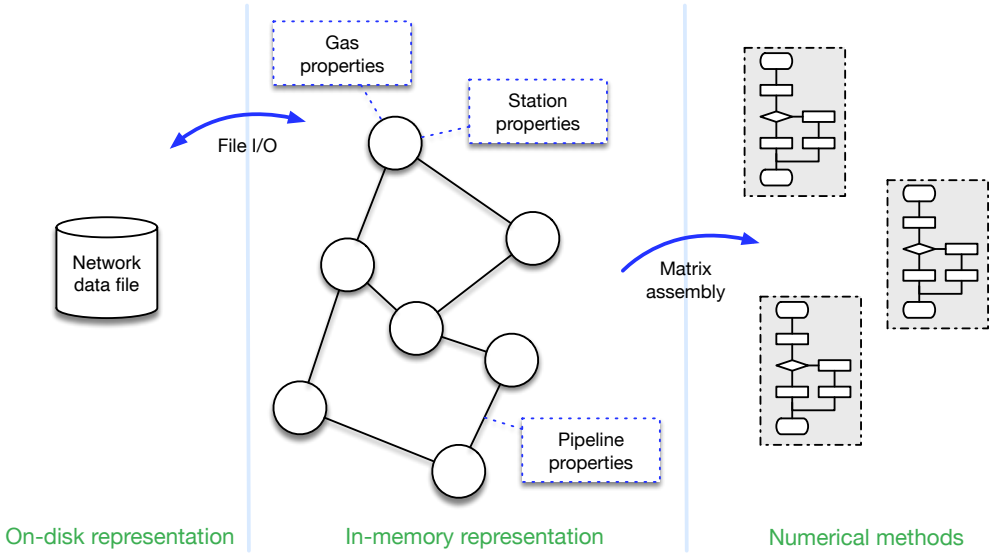


Figure 1: System architecture scheme.

The typical workflow of a Shimmer++ session is as following:

- Create an empty Network Data File (NDF) based on the Shimmer++ database schema
- Populate the NDF with the network data (either manually or from other codes)
- Provide a small configuration file with the simulation parameters
- Run the tool
- Read the results from the NDF and postprocess them with the appropriate software

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

3 Installation

The repository at `https://github.com/shimmerhydrogen/shimmer` includes the Shimmer++ sources, the original matlab code on which Shimmer++ is based (it is not a direct translation) and some documents related to the underlying physical model, along with the github issues that record the developments of Shimmer++.

3.1 Preparing a Linux environment

As first step you should make sure that your system is up-to-date. For Debian-based operating systems:

```
sudo apt update
```

The next step is to install the dependencies which are SQLite, Eigen and the Boost Graph Library (BGL). SQLite is a file-based relational database used to store the gas networks, Eigen is concerned with the linear algebra and the BGL is a graph library used to represent the networks in-memory. The development environment for Shimmer++ is based on the CMake build system and GCC or Clang compilers. Shimmer++ is written in C++20, therefore a relatively recent compiler is needed. To install the dependencies on Debian-based operating systems:

```
sudo apt install -y make cmake build-essential git ctest
sudo apt install -y libsqlite3-dev lua5.4
sudo apt install -y libeigen3-dev
sudo apt install -y libboost-graph-dev libboost-dev
```

3.2 Preparing a Windows environment

Shimmer++ does work on Windows, however it is not the preferential platform. On Windows, Shimmer++ can be compiled via MSYS2, available for download at `https://www.msys2.org/`.

Once you have MSYS2 set up, launch the "MSYS2 UCRT64" shell and install the Shimmer++ prerequisites:

```
pacman -S --needed base-devel
pacman -S --needed mingw-w64-ucrt-x86_64-toolchain
pacman -S --needed mingw-w64-ucrt-x86_64-boost
pacman -S --needed mingw-w64-ucrt-x86_64-lua
pacman -S --needed mingw-w64-ucrt-x86_64-eigen3
pacman -S --needed mingw-w64-ucrt-x86_64-make
pacman -S --needed mingw-w64-ucrt-x86_64-cmake
pacman -S --needed mingw-w64-ucrt-x86_64-ninja
pacman -S --needed git sqlite
```

3.3 Building shimmer++

You can now clone the shimmer++ repository using

```
git clone https://github.com/shimmerhydrogen/shimmer.git
```

or, if you have access to edit:

```
git clone git@github.com:shimmerhydrogen/shimmer.git
```

Some submodules need to be installed. To do that run:

```
git submodule init
git submodule update
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

Now you need to change directory and go into the directory `shimmer++`

```
cd shimmer/shimmer++
```

To build Shimmer++ we advise to create the `build` directory inside your working copy and build Shimmer++ from there:

```
mkdir build
```

enter inside the directory build

```
cd build
```

and then run cmake to create the CMake files

```
cmake ..
```

and finally build the code using

```
cmake --build .
```

The commands above will produce, among others, an executable named `shimmer_solver` in the directory `shimmmer/shimmer++/build`.

Shimmer++ can be installed in the system by running

```
make install
```

from the build directory. By default, Shimmer++ is installed in `/usr/local`, in particular the solver executable will be installed in `/usr/local/bin`, the libraries in `/usr/local/lib` and the database schema plus an example configuration file will be placed in `/usr/local/share`. The installation prefix can be changed by setting appropriately the `CMAKE_INSTALL_PREFIX` variable when running CMake.

3.4 Running Shimmer++

Shimmer++ input/output is done via NDFs, which are SQLite databases with a specific structure. The database schema employed by the NDFs is provided in the source tree at `shimmer/shimmer++/share/shimmer.sql`.

The first step to interact with Shimmer++ is therefore initializing appropriately a NDFs, and this can be done in two ways.

The first way is to use directly the Shimmer++ solver as following:

```
shimmer_solver --init-db newndf.db
```

The last command initializes an empty NDF file named `newndf.db`.

The second way is to use directly the `sqlite3` tool and the provided `shimmer.sql` file. Therefore, assuming that you want to create a new, empty NDF named `newndf.db` and that you have the NDF schema file `shimmer.sql` in the current folder, to initialize a new database:

```
sqlite3 newndf.db < shimmer.sql
```

Remark. *Once the NDF is created, it remains empty. The user must fill in all relevant data tables before running the simulation. The NDF can be populated either manually using an SQLite graphical frontend (e.g., SQLite Browser, `https://sqlitebrowser.org/`), directly through SQL statements with the `sqlite3` tool, or programmatically via different programming languages. Manual manipulation of the database is strongly discouraged, as it is a tedious and error-prone process. Except for debug situations, interaction with NDFs should always happen via custom scripts or third party tools. In the `shimmer++` git repository, we provide an example of a simple MATLAB tool that populates the NDF from a MATLAB graph network. In addition, in `shimmer/shimmer++/src/utils/dbscripts` some Python script to aid filling the NDFs are provided. See Section 5.5 for more details.*

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

Remark. *The file* `shimmer.sql` *must be present in* `${CMAKE_INSTALL_PREFIX}/share`*, in the current work directory or in a subdirectory named* `sqlite` *inside the current work directory. Alternatively, it is possible to specify the fully-qualified path of* `shimmer.sql` *via the* `SHIMMER_SCHEMA_FILE` *environment variable.*

Once the database is ready, a small configuration file is needed. Shimmer++ configuration files are written in the Lua programming language. For example:

```
config.database = "newndf.db"    -- Path of the NDF
config.steps = 7                 -- Timesteps of the simulation
config.dt = 3600                 -- delta-t between timesteps
config.dx = 300e3                -- pipe refinement step
config.tol = 1e-4                -- fluid-dynamic solver tolerance - unsteady cycle
config.tol_std = 1e-14           -- fluid-dynamic solver tolerance - steady cycle
config.refine = false            -- refine pipes
config.quality_tracking = false -- Use quality tracking solver
config.qt_steady= false          -- Do unsteady quality tracking run
```

The variables `config.refine`, `config.quality_tracking`, and `config.qt_steady` are set to `false` by default, so it is not necessary to include them if they are not used in the setup. The variable `config.qt_steady` is only used in case `config.quality_tracking` = `true`.

We save the configuration in a file named, for example, `config.lua`. Shimmer++ is then run as

```
./shimmer_solver config.lua
```

3.5 Shimmer++ organization

The Shimmer++ code is mainly organised with sources stored in the `shimmer/shimmer++/src` folder and unitary tests in the `shimmer/shimmer++/unit_tests`. There is an additionally GERG folder which accounts for the interface of shimmer++ with the code regarding GERG-2008 equation computations.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

4 Model description

Natural Gas infrastructure, being a set of physically interconnected elements forming a network structure, can be represented with a graph. In mathematical term, a graph is an ordered pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of elements called vertices (or nodes) and $\mathcal{E}$ an ordered set of vertex pairs called directed edges (or directed branch). Therefore, in a physical network representation, a directed edge represents any element of the network connecting an inlet and an outlet node with a defined direction. With such a schematization, the fluid-dynamic variables are associated to topological entities in this way:

- nodal: pressure (p), exchanged mass flow rate with external environment ($\dot{m}_{ext}$, G_{ext} or L), gas composition (molar fraction);
- branch: mass flow rate ($\dot{m}$ or G).

To be able to consider the variety of elements in a real gas network, nodes and branches shall be categorized in different types. The following two paragraphs are devoted to the description of the possible types of *nodal stations* and of *branch elements*. Subsections 4.1 and 4.2 are meant to describe the main implemented stations, items of the gas network which requires specifications such as set-points, operational limits etc. which acts like boundary conditions (with their profile in time if they are changing in time) and the technical parameters (such as diameters, length, heigh, etc.). When *quality tracking and admixing* is to be performed, among the boundary conditions the specification of the nodal composition in the gas entry points acts as the boundary condition for the quality tracking problem. The reader is then referred to the subsequent section (Sect. 5) where the database structure for data input/output is described. In this way, the structure of each table would be easily understood.

Subsection 4.3 briefly explains the fluid-dynamic solver part of the overall gas network model, which is also addressed more in-depth in Sect. 6.3. Overall, the model is summarized in the flow chart of Fig. 2. It is possible to note that, after the data acquisition and initialization phase, the time cycle starts. For each timestep, the gas quality is defined and an iterative procedure is followed to solve the linearized fluid-problem. Once a first solution is reached, the verification of the constraints posed on relevant nodal stations and branch element is performed. If one of these items displays a violation of the constraints, then the station state (e.g. control mode) is changed and the fluid-dynamic problem should be re-calculated from scratch, as some of the boundary conditions have been changed. When an acceptable solution is reached, the fluid network has been solved (i.e. all the nodal pressures, the pipeline mass flow rates and the nodal inward/outward mass flow rates have been determined). This result allows to address the quality tracking and admixing box: along the pipeline, the gas quality movement is tracked according to the velocities calculated from the step before, thus a spatial discretization is desired. At mixing nodes (i.e. when two pipelines converge in one node), the gas admixing is simulated, solving a mass balance equation for each chemical specie (the model can handle up to 21 chemical species, as the GERG-2008 equation of state is implemented). When the quality box calculation have been terminated, the time counter is updated and a new time cycle is addressed, in which the gas quality is updated according to the result of the step before and the old solution of pressure and mass flow rate is used as representative of the previous time step.

The model have been consturcted such that it is possible to choose whether to perform an unsteady (transient) simulation or a seady-state one, with or without quality tracking. In case of steady state with quality tracking, only the "admixing" procedure is possible: the result will then be representative of an equilibrium state, even from the point of view of the gas quality distribution.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

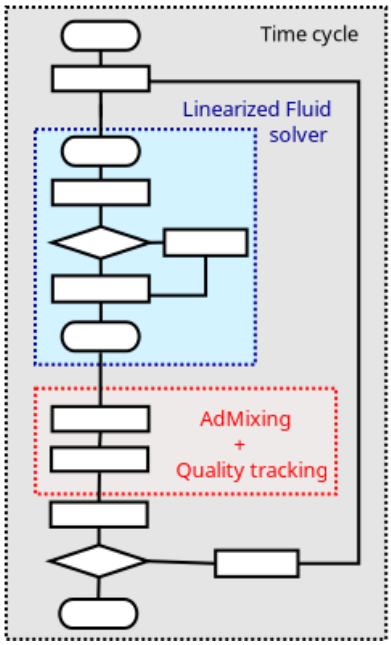


Figure 2: Overall gas network model flow chart.

4.1 Nodal station types

The nodes are generally the interfaces between the gas network and the outer world. For this reason, these are the elements on which to assign the boundary conditions. The quantities which can be assigned are, alternatively, the nodal pressure p_n or the nodal mass flow rate L_n exchanged with the external environment. The convention of the sign for the nodal mass flow rate L_n is as follows

- exiting nodal mass flow rate (consumption) $\rightarrow L_n > 0$
- entering nodal mass flow rate (injection) $\rightarrow L_n < 0$.

Each node station is topologically and unambiguously defined by a node number n. Additionally, localization information can also be added such as: latitude and longitude coordinates and the altitude. While the first two information does not enter into the fluid-dynamic model (they might be useful for postprocessing of results) the altitude is in fact used to take into account the gravitational effect on the pressure drop calculation within the pipelines.

Given that, in general, the simulation is of an unsteady state, the boundary conditions may change over time either in value and in the controlled variable itself, defining the "state" of the nodal station. In the following, a list of the available nodal station, their control states and the related boundary conditions are given. All the limits that, if surpassed, cause a switch in the control mode (thus a switch in state) are referred to as *"Hard Limits"*. *"Soft Limits"* are instead intended to be the normal or desired working conditions. When they are surpassed, a warning message is given, but no further actions are automatically taken from the model.

4.1.1 Pressure regulated entry station without backflow

Pressure regulated entry points can be associated with city-gate gas entry points or with llower-levelgas reduction stations as well as the gas entry point of a transmission system. In all these cases, these items are devoted to maintaining a fixed pressure setpoint at the relevant node. When dealing with the modelling of old types of reduction stations, the pressure set-point is fixed and constant throughout the whole simulation period. If a modulating-pressure reduction station is to be modelled, then the pressure boundary condition will be a time series of predefined pressure set-points. When providing a pressure set point to a node of the

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

network, the gas flow rate will be calculated as a consequence of the fluid-dynamic of the network in order to respect the constraint set by the boundary condition.

This pressure regulated station is also referred to as ReMi station w/o backflow (ReMi is the Italian acronym for the Regulation and Metering stations, usually at the distribution network entry gates). They are usually controlled in pressure, but their architecture is such that in case of any overpressure in the downstream portion of the network (meaning that the pressure on the distribution system side is higher than the pressure set-point), the gas flux is stopped.

This pressure regulated station has as internal hard limit the following condition

$$L_n(t) \leq 0. \tag{1}$$

If it is not respected, then the control mode switches from pressure based to gas flow based and becomes

$$L_n(t) = 0, \tag{2}$$

so to forbid any reverse gas flow. Through these set of boundary condition a typical Re-Mi gas station or second level gas reduction station is modelled: these items in fact are meant to reduce pressure from higher pressure level to distribution range ones, by setting an outlet pressure set point. Nevertheless, this kind of station can also be used for transmission system applications, whenever a pressure regulated control entry point is needed. In Fig. 4, the graphical flowchart is given to explain the switch of states.

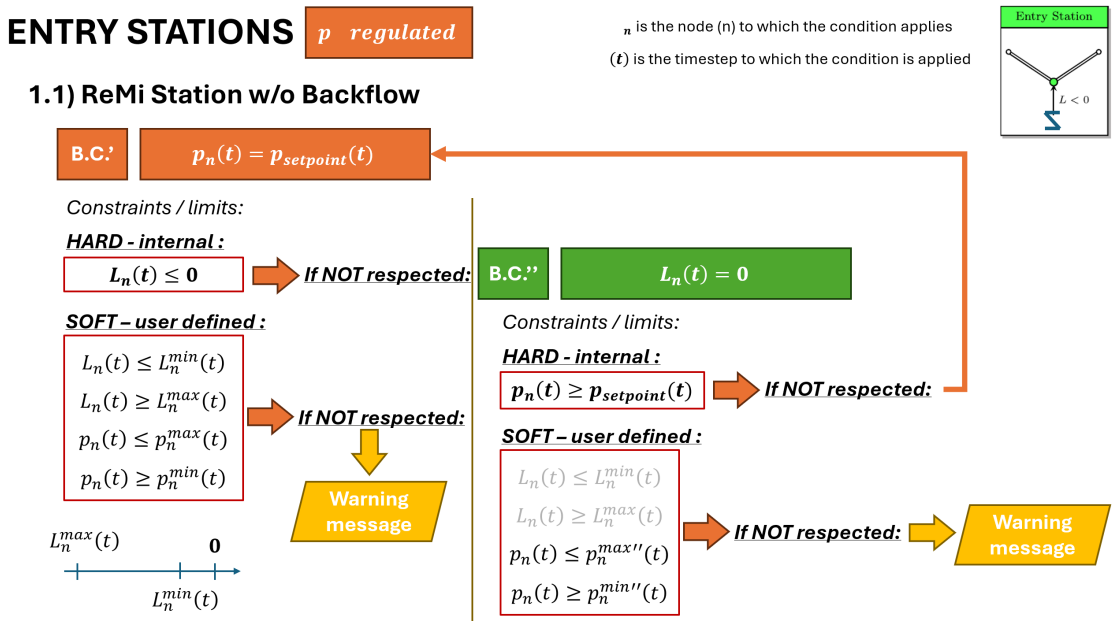


Figure 3: Pressure regulated entry station w/o backflow - flowchart representation.

4.1.2 Mass Flow regulated entry station with pressure control

Flow rate regulated entry points can be associated with gas entry points of the network where a known flux must be delivered: these are, for example, gas production fields, biomethane plants and, in perspective, hydrogen production and injection points (for blending purposes). When providing a flow rate set point to a node of the network, the nodal pressure will be calculated as a consequence of the fluid-dynamic of the network calculated in order to respect the constraint set by the boundary condition.

This mass flow regulated station has as internal hard limit the following condition:

$$p_n(t) \leq p_{\text{threshold}}(t) = f \cdot p_{\text{setpoint}}(t),$$

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

with the parameter $f \in [0, 1]$ representing a *relative threshold factor* with respect to the setpoint value. It defines the maximum admissible fraction of the setpoint that the variable $p_n(t)$ can reach. When $p_n(t)$ exceeds $f \cdot p_{\text{setpoint}}(t)$, the control mode switches from gas flow based to pressure control and becomes:

$$p_n(t) \leq p_{\text{setpoint}}(t).$$

Given a certain gas flow rate forced to enter the gas network, if the reached pressure in the injection point is higher than a certain pressure threshold, it might not be possible for the station or acceptable for the network to receive and then, a pressure set point should be put in force to calculate the allowable entering gas flux. This simulates cases of curtailments of renewable gases from the gas network.

In Fig. 4, the graphical flowchart is given to explain the switch of states.

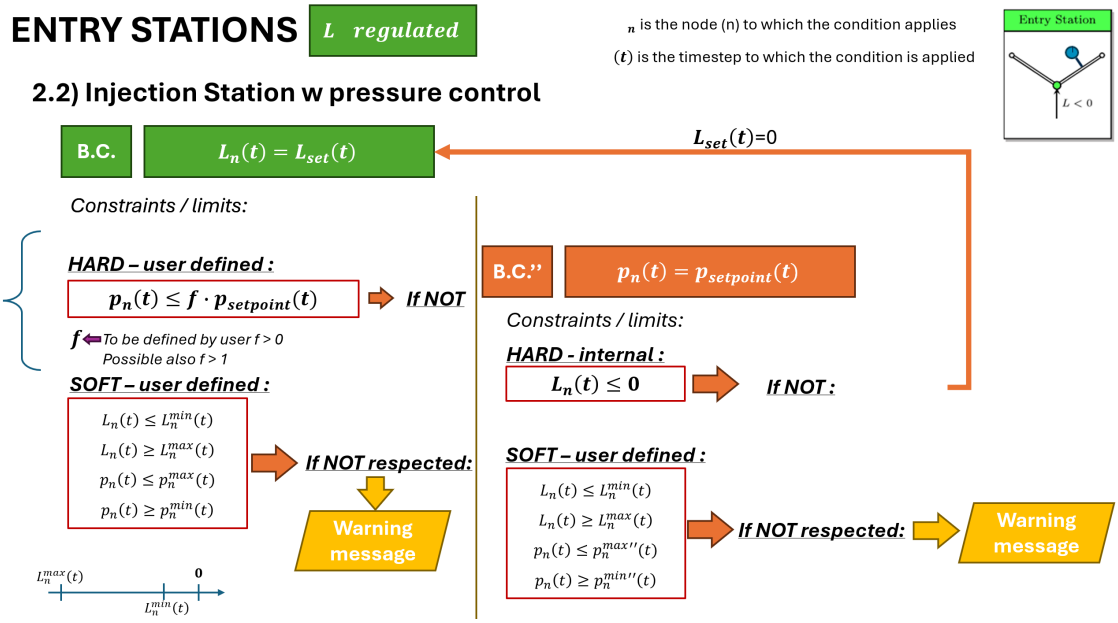


Figure 4: Mass regulated entry station w pressure control - flowchart representation.

4.1.3 Junctions

These are nodes that does not interacts with the external environment: they do not exchange any mass flow. From a boundary condition point of view:

$$L_n(t) = 0 \qquad \text{for any time } t$$

Consequently, the nodal pressure $p_n(t)$ is computed for any timestep. It is to be highlighted that, in the Shimmer++ architecture, once a node is defined as *"junction"*, this boundary conditions are automatically set.

4.1.4 Exit Stations (consumption points)

These nodes are the ones in which a consumption of gas is given. They can be seen as flow rate regulated stations without any control on the resulting pressure. The user should here specify, for each timestep, the set value of consumption, providing the model with the gas consumption profile, with $L_n(t) \geq 0$.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

4.2 Branch element types

The branches are generally the connections between two nodes (either *stations* or *junctions*). Most of the time, they correspond to pipelines as physical entities. However, in the gas network infrastructure, at least two other physical entities can be modeled as branches: compression stations and pressure reduction stations. Basically, the first one is used to increase the pressure, and the second one is used to reduce the pressure. Each branch-like element is topologically defined by

- inlet node $\rightarrow n$ that should be consistent with the overall node numbering
- outlet node $\rightarrow n$ that should be consistent with the overall node numbering

4.2.1 Compression Station

In this modeling framework, the compression station is considered as a black box that contains the combination of all the installed compressors and the gas coolers. In this way, the outlet gas temperature can be assumed the same as the one assumed throughout the system. The general form of the gas compressor equation referred to the mechanical shaft power (P_{shaft}) required by the compressor to the gas turbine driver, results in:

$$P_{\text{shaft}} = \frac{1}{\eta_{is}\,\eta_{mecc}} \frac{\gamma}{\gamma - 1} Z_{in}\, T_{in}\, R_{in} \left(\beta^{\frac{\gamma-1}{\gamma}} - 1\right) \dot{m}_j$$

with η_{is} the isentropic efficiency of compression, η_{mecc} the mechanical efficiency of the compressor (set equal to 80%), γ the adiabatic exponent, and β the compression ratio which is the ratio between outlet and inlet pressure. Z, T, R are respectively the gas compressibility factor, the temperature and the specific gas constant.

Similar to the nodal stations, the compression station needs the specification of a boundary condition that depends on its control mode. The compression station possible control modes are listed in Tab. 1

Controlled variable	Set point	Equation
Driver Power	$P_{\text{shaft setpoint}}$	$P_{\text{shaft setpoint}} = \frac{1}{\eta_{is}\,\eta_{mecc}} \frac{\gamma}{\gamma-1} Z_{in}\, T_{in}\, R_{in} \left(\beta^{\frac{\gamma-1}{\gamma}} - 1\right) \dot{m}_j$
Outlet pressure	$p_{\text{outlet setpoint}}$	$p_o = p_{\text{o,setpoint}}$
Inlet pressure	$p_{\text{inlet setpoint}}$	$p_i = p_{\text{i,setpoint}}$
Pressure Ratio	β_{setpoint}	$\frac{p_o}{p_i} = \beta_{\text{setpoint}}$
Mass flow	$\dot{m}_{\text{setpoint}}$	$\dot{m}_j = \dot{m}_{\text{set}}$

Table 1: Control modes for compressors.

The choice of one of this control mode changes the equation that simulate the compression station branch. Similar to the nodal stations, also for the compression stations some operative ranges of pressure, mass flow, pressure ratio and driver power should be specified as limits and, if violated, they may lead to changes in the control mode.

Furthermore, the gas compression station can also assume two different states: ON and OFF: when the compressor is ON, one of the above-mentioned control mode needs to be specified; when the compressor is OFF, two sub-states are possible (and should be specified):

- BYPASS $\rightarrow \dot{m}_{\text{in}} = \dot{m}_{\text{out}}$; The compressor leaves free-flow thus the inlet and outlet mass (as well as the pressures) are set as equal.
- CLOSED $\rightarrow \dot{m}_{\text{j,compr}} = 0$ The compressor acts like a closed valve. It interrupts the gas flow. This is relevant when the downstream section of the grid is at higher pressure than the upstream (so to avoid counterflows).

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

4.2.2 Pipelines

The pipelines are the most common branch-like items. They are fully passive elements with no boundary conditions requirements. Hence, each single pipe is defined by their technical parameters features, listed hereafter

- Internal Diameter $\rightarrow D_i$
- Length $\rightarrow l$
- Internal Roughness $\rightarrow \varepsilon_i$

In this modelling framework, in principle, the user can define pipelines with any lengths, which can differ a lot one-another. In case the user wants or need to have more uniform spatial discretization, it is possible to define the number of segments ($\#_{\text{segs}}$), for each node. The model will the subdivide the pipeline in the assigned number of segments, dividing uniformly the pipe length by creating a number of so-called "fictitius" nodes, treated as junctions.

The pipeline equation employed in this model is the Ferguson Equation, written as follows

$$P_{in} - P_{out}e^{s_j} = \frac{2\,\bar{p}_j\,l_{e_j}}{A_j}\frac{\partial \dot{m}_j}{\partial t} + \frac{\lambda_j\,\bar{c}_j^2\,l_{e_j}}{D_j A_j^2}\,\dot{m}_j|\dot{m}_j|,$$

with

$$l_{e_j} = \begin{cases} l_j, & h_{in} = h_{out} \\ \dfrac{e^{s_j}-1}{s_j}\,l_j, & h_{in} \neq h_{out} \end{cases} \qquad \text{with } s_j = \frac{2g(h_{out}-h_{in})}{\bar{c}_j^2},$$

where $P = p^2$ is the quadratic pressure (p in $[Pa]$), $\dot{m}_j$ is the gas mass flow rate $[kg/s]$, λ is the friction factor $[-]$, $\bar{c}^2$ is the isothermal speed of sound of the gas $[m/s]$, D_j is the pipeline inner diameter $[m]$, A_j is the pipeline cross section area $[m^2]$, l_j is the pipeline (or pipeline section) length $[m]$, g is the gravitational acceleration $[m/s^2]$ and the subscripts *in* and *out* stand for the inlet and the outlet sections of the generic j^{th} pipe. In order to account for the gravitational contribution, the "effective length" l_e is defined as the corrected length of the pipeline section in case of non-horizontal pipelines (whose slope is defined by the elevation difference $(h_{out} - h_{in})$ of their ends). The averaged quantities, which make the analytical solution possible, are calculated starting from the computed value of the average pressure $\bar{p}$ as

$$\bar{p} = \frac{p_{in}^2 + p_{in}p_{out} + p_{out}^2}{p_{in} + p_{out}},$$

and in turn, the speed of sound is computed using the gas relation

$$\bar{c}^2 = Z(\bar{p}, T, [y])\,RT$$

where $Z(\bar{p}, T, [y])$ is obtained from the application of the chosen gas Equation of State (EoS). In this modelling framework the GERG-2008 EoS has been implemented [1]. Also the friction factor λ can be calculated by means of an appropriate friction factor correlation. In this framework, the Cheng correlation is used [2].

For an in-depth knowledge of the linearization and space-time discretization of the Ferguson formula the reader is invited to refer to [3] [4].

4.3 Overall Network Model Rationale

Solving a fluid networks means to solve a pressure-velocity coupled problem on a graph-modeled infrastructure. In this case, the velocity variable has been replaced by the pipeline mass gas flow rates $\dot{m}_j$. For solving the gas network, two sets of equations are needed: a set of equations for all the branches and a set of equations for all the nodes. In this framework, the branch equations are the set of quadratic pressure drop equation

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

which can be written for all the pipes (Fergusson Equation) and the nodal equations are the set of mass balance (mass conservation law) which can be applied to each nodes. Whenever a non-pipeline station is defined, the pressure drop equation is substituted with the relevant equation describing the element (and its control mode). Besides these, all the relevant boundary conditions are needed, as discribed in the previous sections. By linearizing the pressure drop equation it is possible to assemble an algebraic problem for the full gas network infrastructure, provided that the topological structure of the network (i.e. all the node-branch connections) is properly arranged in an incidence matrix. To have more in-depth insight of this subject the reader is referred to [4]. In 8 a schematic representation of the mathematical procedure is displayed.

4.4 Nomenclature and unit of measurments

In this section, a summary table showing the units of measurement for each variable is given below:

	Input/output variable	**Symbol**	**Unit of measurement**
Nodal variables	Pressure	p	Pa
	Injected/consumed gas	$\dot{m}_{\text{ext}}$ or G_{ext} or L	kg/s
		Injected	*negative value*
		Consumed	*positive value*
	Gas composition	`Frac_XX`	Molar fraction [0–1]
Branch variables	Mass flow	$\dot{m}$ or G	kg/s
		Same direction of the branch	*Positive value*
		Opposite direction of the branch	*Negative value*
	Velocity	v	m/s
		Same direction of the branch	*Positive value*
		Opposite direction of the branch	*Negative value*
	Diameter	D	m
	Length	l	m
	Roughness	ε	m
	Compressor power	P_{cmpr}	W

Table 2: Nodal and branch variables with their symbols and units of measurement.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

5 Network Data Files

The Shimmer++ input/output is done entirely via network data files (NDFs). The user prepares NDFs specifying the whole network and the code processes them and stores simulation solutions in them. The NDFs of Shimmer++ are based on SQLite, which is the most widely deployed relational database system. This choice guarantees total compatibility with any programming language/environment and easy manipulation of the network data via standard SQL queries. On top of the raw SQLite database, Shimmer++ provides a C++ API and a Matlab API specific for the manipulation of the NDFs.

5.1 Database schema: nodal elements *(stations)*

In the following the NDF database schema will be described. The knowledge of the database schema is needed to directly modify the NDFs via tools like SQLite Browser (`https://sqlitebrowser.org/`) or to implement third party tools able to read and write Shimmer++ NDFs.

5.1.1 Station types

Shimmer++ handles gas networks with different types of stations, as described in Section 4.1. Each station type is assigned a specific numeric identifier in order to be appropriately handled by the code. Currently implemented station types are visible in `network_elements.hpp` and summarized in the following code snippet:

```
enum class station_type : int {
    ENTRY_P_REG = 1,            /* ReMi w/o backflow */
    ENTRY_L_REG = 2,            /* Injection w/ pressure control */
    EXIT_L_REG  = 3,            /* Consumption point w/o pressure control */
    JUNCTION    = 4,            /* Junction */
    PRIVATE_INLET  = 10,        /* Inlet, internal use only */
    PRIVATE_OUTLET = 11,        /* Outlet, internal use only */
    FICTITIOUS_JUNCTION = 100   /* For quality tracking, should not appear in DB*/
};
```

On the NDF side, the table containing the station types is named `station_types` and is specified by the following SQL statement:

```
create table station_types (
    t_type          INTEGER,        -- Station type, as per enum above
    t_descr         TEXT NOT NULL,  -- Free-form description of the type
    t_limits_table  TEXT,           -- Table with type-specific operational limits
    t_profile_table TEXT,           -- Table with type-specific operational profiles

    PRIMARY KEY(t_type)
);
```

This table comes pre-populated to reflect the numeric ids cited above and it should not be modified by the user.

t_type	t_descr	t_limits_table	t_profile_table
1	ReMi station w/o backflow	limits_remi_wo	profiles_remi_wo
2	Injection station w/ pressure control	limits_injection_w	profiles_injection_w
3	Consumption point w/o pressure control	limits_conspoint_wo	profiles_conspoint_wo
4	Junction	NULL	NULL
10	Inlet station - private	NULL	NULL
11	Outlet station - private	limits_outlet_priv	profiles_outlet_priv

Table 3: List of possible nodal station types.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

The columns `t_limits_table` and `t_profile_table` contain the table names for the settings of a specific station type. Should the code be modified to support new station types, this table must be updated accordingly.

5.1.2 Stations List

The table `stations` contains the list of all the stations present in a network. A station is uniquely identified by the field `s_number` and has the following attributes:

- `s_name`: the name of the station
- `t_type`: the type of the station, according to the table `station_types`
- `s_height`: altitude of the station
- `s_latitude`: latitude of the station
- `s_longitude`: longitude of the station

The table is fully specified by the following SQL code:

```
create table stations (
    s_number    INTEGER,
    s_name      TEXT NOT NULL,
    t_type      INTEGER,

    s_height    REAL DEFAULT 0.0 NOT NULL,
    s_latitude  REAL DEFAULT 0.0 NOT NULL,
    s_longitude REAL DEFAULT 0.0 NOT NULL,

    PRIMARY KEY(s_number),

    -- The type of the station must be well-defined
    FOREIGN KEY (t_type)
        REFERENCES station_types(t_type),

    CHECK(s_number >= 0)
);
```

5.1.3 Limits and profiles: Pressure regulated entry station w/o backflow *(ReMi station)*

Each station of type *"ReMi"* has an associated set of user-specified limits on pressure and mass flow rate. Limits are specified per station by NDF entries including:

- `s_number`: number of the station
- `lim_Lmin`: minimum allowed mass flow rate (*soft limit*)
- `lim_Lmax`: maximum allowed mass flow rate (*soft limit*)
- `lim_Pmin`: minimum allowed pressure (*soft limit*)
- `lim_Pmax`: maximum allowed pressure (*soft limit*)

As described in Section 4.1, the *hard limit* regarding the avoidance of the *reverse gas flow* is built-in the station control mode and the user is not requested to specify it. The table of the limits for ReMi stations is fully specified by the following SQL code:

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

```
create table limits_remi_wo (
    s_number    INTEGER UNIQUE,
    lim_Lmin    REAL DEFAULT 0.0 NOT NULL,
    lim_Lmax    REAL DEFAULT 0.0 NOT NULL,
    lim_Pmin    REAL DEFAULT 0.0 NOT NULL,
    lim_Pmax    REAL DEFAULT 0.0 NOT NULL,

    FOREIGN KEY (s_number)
        REFERENCES stations(s_number)
);
```

In addition to limits, the NDFs allow to store also profiles for the quantity controlled by the station (the pressure, in this case). A pressure profile is specified by triples including:

- `s_number`: number of the station
- `prf_time`: relative time of the sample (seconds)
- `prf_Pset`: pressure setpoint at the specified time

If a profile for a station includes a single entry, the quantity is held constant at the specified value for the whole simulation. The table of the profiles for ReMi stations is fully specified by the following SQL code:

```
create table profiles_remi_wo (
    s_number    INTEGER,
    prf_time    REAL DEFAULT 0.0 NOT NULL,
    prf_Pset    REAL DEFAULT 0.0 NOT NULL,

    FOREIGN KEY (s_number)
        REFERENCES stations(s_number)
);
```

5.1.4 Limits and profiles: Mass flow regulated entry station w/ pressure control *(Injection station)*

Each station of type *"Injection"* has an associated set of user-specified limits on pressure and mass flow rate. Limits are specified per station by NDF entries including:

- `s_number`: number of the station
- `lim_Lmin`: minimum allowed mass flow rate (*soft limit*)
- `lim_Lmax`: maximum allowed mass flow rate (*soft limit*)
- `lim_Pmin`: minimum allowed pressure (*soft limit*)
- `lim_Pmax`: maximum allowed pressure (*soft limit*)
- `parm_f`: *relative threshold factor* with respect to the pressure setpoint value ($f \in [0, 1]$) - see Section 4.1.

The table of the limits for Injection stations is fully specified by the following SQL code:

```
create table limits_injection_w (
    s_number    INTEGER UNIQUE,
    lim_Lmin    REAL DEFAULT 0.0 NOT NULL,
    lim_Lmax    REAL DEFAULT 0.0 NOT NULL,
    lim_Pmin    REAL DEFAULT 0.0 NOT NULL,
    lim_Pmax    REAL DEFAULT 0.0 NOT NULL,
    parm_f      REAL DEFAULT 1.0 NOT NULL,
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

```
    FOREIGN KEY (s_number)
        REFERENCES stations(s_number)
);
```

In addition to limits, the NDFs allow to store also profiles for the quantity controlled by the station. A profile for Injection stations is specified by quadruples including:

- `s_number`: number of the station
- `prf_time`: relative time of the sample (seconds)
- `prf_Pset`: pressure setpoint at the specified time (*hard limit* and secondary set-point in case of change of control mode)
- `prf_Lset`: mass flow rate setpoint at the specified time - the primary set-point of the station: injected mass flow rate.

If a profile for a station includes a single entry, the quantity is held constant at the specified value for the whole simulation. The table of the profiles for Injection stations is fully specified by the following SQL code:

```
create table profiles_injection_w (
    s_number    INTEGER,
    prf_time    REAL DEFAULT 0.0 NOT NULL,
    prf_Pset    REAL DEFAULT 0.0 NOT NULL,
    prf_Lset    REAL DEFAULT 0.0 NOT NULL,

    FOREIGN KEY (s_number)
        REFERENCES stations(s_number)
);
```

5.1.5 Limits and profiles: Consumption station

Each station of type Consumption has an associated set of user-specified limits on pressure and mass flow rate. Limits are specified per station by NDF entries including:

- `s_number`: number of the station
- `lim_Lmin`: minimum allowed mass flow rate
- `lim_Lmax`: maximum allowed mass flow rate
- `lim_Pmin`: minimum allowed pressure
- `lim_Pmax`: maximum allowed pressure

The table of the limits for Consumption stations is fully specified by the following SQL code:

```
create table limits_conspoint_wo (
    s_number    INTEGER UNIQUE,
    lim_Lmin    REAL DEFAULT 0.0 NOT NULL,
    lim_Lmax    REAL DEFAULT 0.0 NOT NULL,
    lim_Pmin    REAL DEFAULT 0.0 NOT NULL,
    lim_Pmax    REAL DEFAULT 0.0 NOT NULL,

    FOREIGN KEY (s_number)
        REFERENCES stations(s_number)
);
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

In addition to limits, the NDFs allow to store also profiles for the quantity controlled by the station. A mass flow rate profile is specified by triples including:

- `s_number`: number of the station
- `prf_time`: relative time of the sample (seconds)
- `prf_Lset`: mass flow rate setpoint at the specified time

If a profile for a station includes a single entry, the quantity is held constant at the specified value for the whole simulation. The table of the profiles for Consumption stations is fully specified by the following SQL code:

```
create table profiles_conspoint_wo (
    s_number    INTEGER,
    prf_time    REAL DEFAULT 0.0 NOT NULL,
    prf_Lset    REAL DEFAULT 0.0 NOT NULL,

    CHECK(prf_Lset >= 0),

    FOREIGN KEY (s_number)
        REFERENCES stations(s_number)
);
```

5.1.6 Gases

The `gases` table lists the gases supported by the Equation of state described in Section 6.3.3. They can be employed in simulations, either as pure substances or as components of mixtures. Each gas is identified by thefollowing attributes

- `g_num`: number id of the gas
- `g_formula`: as the name indicates, it corresponds to the gas formula (in capital letters)
- `g_name`: the name of the corresponding gas

The table is fully specified by the following SQL code

```
create table gases (
    g_num       INTEGER PRIMARY KEY,
    g_formula   TEXT NOT NULL,
    g_name      TEXT NOT NULL
);
```

This table comes pre-populated to reflect the numeric IDs listed below and should not be modified by the user.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

g_num	g_formula	g_name
0	CH4	Methane
1	N2	Nitrogen
2	CO2	Carbon dioxide
3	C2H6	Ethane
4	C3H8	Propane
5	i_C4H10	i-butane
6	n_C4H10	n-butane
7	i_C5H12	i-pentane
8	n_C5H12	n-pentane
9	C6H14	Hexane
10	C7H16	Heptane
11	C8H18	Octane
12	C9H20	Nonane
13	C10H22	Decane
14	H2	Hydrogen
15	O2	Oxygen
16	CO	Carbon oxide
17	H2O	Water
18	H2S	Hydrogen sulfide
19	He	Helium
20	Ar	Argon

Table 4: List of gases with their formula and corresponding names.

5.1.7 Gas molar fractions

The table `gas_molar_fraction` contains the molar fraction of each gas species at a given station (node) in the network. Each record identifies a specific gas component within a node and is uniquely identified by the combination of `s_number` and `frac_<gas_name>`, denoting the following attributes:

- `s_number`: identifier of the station (node)
- `frac_<gas_name>`: molar fraction of each gas species (between 0 and 1), where `<gas_name>` refers to the abbreviations listed in the `gases` table (e.g., `frac_CH4`, `frac_CO2`, `frac_H2`, etc.) shown in the previous section.

The specification of the molar composition for a gas entry node (pressure regulated node or injection node) allows for the simulation of a multi-gas scenario where different gas sources can have different gas composition. Thus, a quality-tracking based simulation can be consequently performed. In other words, it is here that the user should define the composition of the injected gas which is to be tracked.

This setting is used for both the table `gas_molar_fraction` and `solution_gas_molarfrac`: the first stores the initial gas composition, while the latter stores the updated concentrations throughout the simulation. As for other variables output, in the `solution_gas_molarfrac`, there will be an additional column regarding the `timestep`.

```
create table gas_molar_fraction (
    s_number     INTEGER NOT NULL,

    frac_CH4     REAL DEFAULT 0.0 NOT NULL,
    frac_N2      REAL DEFAULT 0.0 NOT NULL,
    frac_CO2     REAL DEFAULT 0.0 NOT NULL,
    frac_C2H6    REAL DEFAULT 0.0 NOT NULL,
    frac_C3H8    REAL DEFAULT 0.0 NOT NULL,
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

```
    frac_i_C4H10 REAL DEFAULT 0.0 NOT NULL,
    frac_n_C4H10 REAL DEFAULT 0.0 NOT NULL,
    frac_i_C5H12 REAL DEFAULT 0.0 NOT NULL,
    frac_n_C5H12 REAL DEFAULT 0.0 NOT NULL,
    frac_C6H14   REAL DEFAULT 0.0 NOT NULL,
    frac_C7H16   REAL DEFAULT 0.0 NOT NULL,
    frac_C8H18   REAL DEFAULT 0.0 NOT NULL,
    frac_C9H20   REAL DEFAULT 0.0 NOT NULL,
    frac_C10H22  REAL DEFAULT 0.0 NOT NULL,
    frac_H2      REAL DEFAULT 0.0 NOT NULL,
    frac_O2      REAL DEFAULT 0.0 NOT NULL,
    frac_CO      REAL DEFAULT 0.0 NOT NULL,
    frac_H2O     REAL DEFAULT 0.0 NOT NULL,
    frac_H2S     REAL DEFAULT 0.0 NOT NULL,
    frac_He      REAL DEFAULT 0.0 NOT NULL,
    frac_Ar      REAL DEFAULT 0.0 NOT NULL,

    PRIMARY KEY (s_number),

    -- Each row must correspond to an existing station
    FOREIGN KEY (s_number)
        REFERENCES stations(s_number),

    CHECK (
        frac_CH4    BETWEEN 0.0 AND 1.0 AND
        frac_N2     BETWEEN 0.0 AND 1.0 AND
        frac_CO2    BETWEEN 0.0 AND 1.0 AND
        frac_C2H6   BETWEEN 0.0 AND 1.0 AND
        frac_C3H8   BETWEEN 0.0 AND 1.0 AND
        frac_H2     BETWEEN 0.0 AND 1.0
        -- ... (additional checks can be added for all gases)
    )
);
```

The sum of all `molar_fraction` values associated with the same `s_number` must be equal to 1.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

5.2 Database schema: branch elements *(pipelines and non-pipe elements)*

5.2.1 Branch element types

All the network elements with an inlet and an outled are termed *"pipeline elements"* inside Shimmer++ (indeed they are branch elements comprising pipeline and non-pipeline elements). As for the stations, Shimmer++ handles different network elements, each uniquely identified by an integer. Pipeline types are contained in a table fully specified by the following SQL code:

```
create table pipeline_types (
    p_type      INTEGER,
    t_name      TEXT NOT NULL,
    PRIMARY KEY (p_type)
);
```

As for the stations, the above SQL table is pre-populated with the four types of stations supported by Shimmer++, summarized in Table 5.

p_type	t_name
0	Plain pipe
1	Compressor
2	Reduction station
3	Valve

Table 5: Pipe-like elements supported by Shimmer++

Each pipeline-like element is uniquely identified by a triple including the pipe name `p_name`, the originating station `s_from` and the destination station `s_to`. In addition, the type of the pipeline element is stored in the `p_type` attribute.

```
create table pipelines (
    p_name      TEXT NOT NULL,
    s_from      INTEGER NOT NULL,
    s_to        INTEGER NOT NULL,
    p_type      INTEGER NOT NULL,
    PRIMARY KEY (p_name, s_from, s_to),

    -- The source station must exist
    FOREIGN KEY (s_from)
        REFERENCES stations(s_number),
    -- The destination station must exist
    FOREIGN KEY (s_to)
        REFERENCES stations(s_number),
    -- The pipeline type must be valid
    FOREIGN KEY (p_type)
        REFERENCES pipeline_types(p_type)
);
```

5.2.2 Plain pipes

Plain pipes are uniquely identified by a triple including the pipe name `p_name`, the originating station `s_from` and the destination station `s_to`. Additional attributes of a pipe are its diameter, its length, its roughness and the number of segments in which it will be split if mesh refinement is used for quality tracking. The table where the pipes and related parameters are stored is fully specified by the following SQL code:

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

```
create table pipe_parameters (
    p_name      TEXT NOT NULL,
    s_from      INTEGER,
    s_to        INTEGER,
    diameter    REAL DEFAULT 0.0,
    length      REAL DEFAULT 0.0,
    roughness   REAL DEFAULT 0.0,
    ref_nsegs   INTEGER DEFAULT 0,

    -- The referenced pipeline must exist
    FOREIGN KEY (p_name, s_from, s_to)
        REFERENCES pipelines(p_name, s_from, s_to),

    CHECK(ref_nsegs >= 0)
);
```

The `ref_nsegs` variable here specifies the refinement parameter when quality tracking is employed. If the value is zero, the pipe is refined according the `config.dx` parameter, in particular the segments will be at most `config.dx` long. Otherwise, it specifies the exact number of segments in which the pipe has to be refined.

5.2.3 Compressor stations

Compressors are uniquely identified by a triple including the pipe name `p_name`, the originating station `s_from` and the destination station `s_to`. Each compressor has an associated set of limits and an associated temporal profile. The physical/engineering meaning of these parameters are described in 4.2.

The table containing the compressor limits is fully specified by the following SQL code:

```
create table compressor_limits (
    p_name          TEXT NOT NULL,
    s_from          INTEGER,
    s_to            INTEGER,

    max_power       REAL DEFAULT 0.0,
    max_outpress    REAL DEFAULT 0.0,
    min_inpress     REAL DEFAULT 0.0,
    max_ratio       REAL DEFAULT 0.0,
    min_ratio       REAL DEFAULT 0.0,
    max_massflow    REAL DEFAULT 0.0,

    PRIMARY KEY (p_name, s_from, s_to),

    -- The referenced pipeline must exist
    FOREIGN KEY (p_name, s_from, s_to)
        REFERENCES pipelines(p_name, s_from, s_to)
);
```

where the control mode is an integer defined following Table 6.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

Name	controlmode	Active	Description
ON_POWER	0	On	Control mode power driver
ON_OPRESS	1	On	Control mode outlet pressure
ON_IPRESS	2	On	Control mode inlet pressure
ON_RATIO	3	On	Control mode compression ratio
ON_MASSFLOW	4	On	Control mode mass flow
OFF_BYPASS	10	Off	Bypass
OFF_CLOSED	11	Off	Closed

Table 6: Control modes supported by Shimmer++ for the compressor station.

Each compressor also has an associated CONSTANT temporal profile.

```
create table compressor_profile (
    p_name      TEXT NOT NULL,
    s_from      INTEGER,
    s_to        INTEGER,

    prf_time    REAL DEFAULT 0.0,
    controlmode INTEGER DEFAULT 10, -- default OFF BYPASS
    power       REAL DEFAULT 0.0,
    outpress    REAL DEFAULT 0.0,
    inpress     REAL DEFAULT 0.0,
    ratio       REAL DEFAULT 0.0,
    massflow    REAL DEFAULT 0.0,

    -- The referenced pipeline must exist
    FOREIGN KEY (p_name, s_from, s_to)
        REFERENCES pipelines(p_name, s_from, s_to)
);
```

5.3 Database schema: network initial conditions

To start a simulation the user must provide the initial **guessed** conditions of the network, especially for the nodes and pipes in which no boundary conditions are given. Thus, the conditions at each node (guessed pressures) and in each pipe (guessed mass flow) are required.

5.3.1 Initial conditions for nodes

The initial conditions for the stations are specified in the table `station_initial_conditions` as specified by the following SQL code:

```
create table station_initial_conditions (
    s_number    INTEGER UNIQUE NOT NULL,
    init_P      REAL DEFAULT 0.0 NOT NULL,
    init_L      REAL DEFAULT 0.0 NOT NULL,

    FOREIGN KEY (s_number)
        REFERENCES stations(s_number)
);
```

For each station, an entry composed of the station number, the station pressure, and the station mass flow exchanged with the external environment must be provided.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

5.3.2 Initial conditions for branches

Similarly to stations, branches initial conditions must also be provided as entries in the `pipe_initial_conditions` table. Each entry has to include the station name, the start and end node and the initial value of guessed mass flow G, as specified by the following SQL code.

```
create table pipe_initial_conditions (
    p_name      TEXT NOT NULL,
    s_from      INTEGER,
    s_to        INTEGER,
    init_G      REAL DEFAULT 0.0 NOT NULL,

    PRIMARY KEY (p_name, s_from, s_to),

    FOREIGN KEY (p_name, s_from, s_to)
        REFERENCES pipelines(p_name, s_from, s_to)
);
```

5.4 Database schema: simulation outputs

The solution of the model implemented by Shimmer++ is stored in five separate tables of the Network Data Files.

Remark. *If the* `pipe refinement` *option is enabled, Shimmer++ automatically generates a new Network Database (NDB) within the* `build` *directory. The file will be named* `refined_<database_name>.db`*, and it will contain also the updated information on* ***stations*** *and* ***pipes*** *corresponding to the additional units created during the refinement process.*

The table `solution_station_pressures` stores for each row the pressure at a given time and a given station.

```
create table solution_station_pressures (
    s_number        INTEGER NOT NULL,
    timestep        INTEGER NOT NULL,
    pressure        REAL DEFAULT 0.0 NOT NULL,

    FOREIGN KEY (s_number)
        REFERENCES stations(s_number)
);
```

The table `solution_pipe_flowrates` stores for each row the flow rate in the pipe identified by `p_name`, `s_from`, `s_to` at a specific timestep.

```
create table solution_pipe_flowrates (
    p_name      TEXT NOT NULL,
    s_from      INTEGER NOT NULL,
    s_to        INTEGER NOT NULL,
    timestep    INTEGER NOT NULL,
    flowrate    REAL DEFAULT 0.0 NOT NULL,

    FOREIGN KEY (p_name, s_from, s_to)
        REFERENCES pipelines(p_name, s_from, s_to)
);
```

The table `solution_station_flowrates` stores for each row the flow rate in the node identified by `s_number` at a specific timestep.

```
create table solution_station_flowrates (
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

```
    s_number        INTEGER NOT NULL,
    timestep        INTEGER NOT NULL,
    pressure        REAL DEFAULT 0.0 NOT NULL,

    FOREIGN KEY (s_number)
        REFERENCES stations(s_number)
);
```

The table `solution_pipe_velocities` stores for each row the gas velocity in the pipe identified by `p_name`, `s_from`, `s_to` at a specific timestep.

```
create table solution_pipe_velocities (
    p_name      TEXT NOT NULL,
    s_from      INTEGER NOT NULL,
    s_to        INTEGER NOT NULL,
    timestep    INTEGER NOT NULL,
    velocity    REAL DEFAULT 0.0 NOT NULL,

    FOREIGN KEY (p_name, s_from, s_to)
        REFERENCES pipelines(p_name, s_from, s_to)
);
```

Finally, the table `solution_station_molarfrac` stores for each row the molar fraction for each `g_name` component of the gas mixture in the node identified by `s_number` at a specific timestep.

```
create table solution_station_molfrac (
    s_number      TEXT NOT NULL,
    timestep      INTEGER NOT NULL,
    g_name        INTEGER NOT NULL,
    molarfrac     REAL DEFAULT 0.0 NOT NULL,

    FOREIGN KEY (s_number)
        REFERENCES stations(s_number)
);
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

5.5 Automatic NDF Creation: sample code in Matlab environment

After the NDF is created, see Section 5 for more details, it remains empty. The user is required to populate it with the relevant data prior to executing the simulation. This section presents a simple MATLAB tool capable of populating the NDF from a MATLAB graph network, as well as generating the network back from the NDF.

Remark. *The reader should consider this file as templates for the construction of further codes based on other programming languages for the database population procedure*

```
%% DB schema path
db_schema = fullfile(pwd, "../../shimmer++/share/shimmer.sql");

%% Load NDF Matlab
graph_path = fullfile(pwd, "graph_example.mat");
graph = load(graph_path);
graph = graph.graph;

%% Load NDF SQlite
db_path = fullfile(pwd, "/graph_example.db");
if exist(db_path, 'file') == 2
    delete(db_path);
end

%% Fill NDF from graph Matlab to SQlite
sql_create(db_path, db_schema);
sql_populate_from_graph(db_path, graph);

%% Fill NDF from SQlite to Matlab
converted_graph = graph_populate_from_sql(db_path);
```

Listing 1: Example of NDF creation from Matlab code `sqlite_populate_graph_example.m`

Listing 1 shows the functions to be called for generating an NDF SQLite file from MATLAB code and viceversa. The complete example is available in the `matlab/sqlite` directory of the `shimmer++` Git repository.

Remark. *MATLAB version R2024b or later is required to run all the codes.*

In the example, we propose the creation of a NDF made by 5 stations and 5 pipes, see Figure 5.

The MATLAB function `sql_create` is used to create a new NDF file at the location specified by `db_path`, based on the SQLite schema provided at `db_schema`. Once the NDF file has been created, it can be populated using the function `sql_populate_from_graph`. This function fills the SQLite database using the MATLAB structure `graph`, which consists of the following fields:

- `Nodes`: a structure containing NDF station information, such as station types (`Types`), boundary and initial conditions (`PRESSURES`, `G_EXE`, `Pset_bc` and `Lset_bc`), and initial and boundary gas molar compositions (`YYc` and `YY_ext_in`);
- `Edges`: a structure containing NDF pipe information, including pipe endpoints (`EndNodes`), pipe description (`Length`, `Epsi`, and `Diameter`) , and initial flow conditions (`FLOWRATES`).

An example of the MATLAB NDF structure is provided in the file `graph_example.mat`.

The inverse operation, that is converting the SQLite NDF database back into the corresponding MATLAB structure, is performed using the MATLAB function `graph_populate_from_sql`. This function takes as input the path to the SQLite database and generates the corresponding MATLAB structure, as shown in the last line of Code 1.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

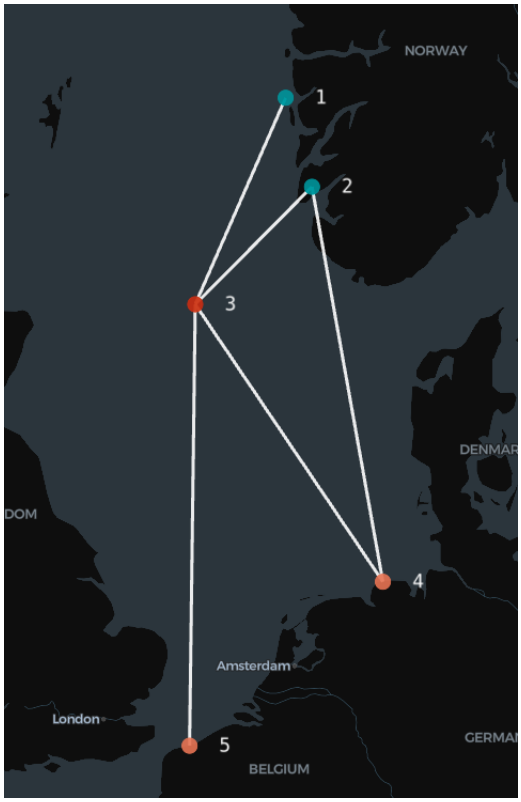


Figure 5: Graph example.

Remark. *If the user is creating an NDF file in this way and attempting to read it through the desktop version of DB Browser for SQLite, it is suggested to update/refresh the database structure after uploading of the .db file, as otherwise it might not be possible to see the actual content of the file.*

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

6 Developer zone

6.1 In memory representation

Network modelling can be mathematically abstracted into a graph of stations (graph nodes) and pipes an non-pipes network elements (graph edges, also called *branches*). The network graph is implemented using Boost Graph Library, which provides a labeled graph implementation and related algorithms. The network data in particular is stored into the graph labels. Shimmer++ represents the network as an undirected graph, therefore pipes are inherently not directional, in contrast to physical variables as velocity or flux. Using the facilities of the Boost Graph Library, the definition of the graph type is

```
using infrastructure_graph = adjacency_list< listS, vecS,
                                undirectedS, vertex_properties, edge_properties>;
```

which is found in `shimmer++/src/infrastructure_graph.h`. In turn, the graph labels have the following types:

```
using namespace boost;

using infrastructure_graph = adjacency_list< listS, vecS,
                                undirectedS, vertex_properties, edge_properties>;
using sptr_pipe_t = std::shared_ptr<edge_station::station>;
using uptr_node_t = std::unique_ptr<station>;

struct vertex_properties
{
   int             u_snum;
   int             i_snum;
   station_type    type;
   double          height;
   double          latitude;
   double          longitude;
   vector_t        gas_mixture;
   uptr_node_t     node_station;

};

struct edge_properties
{
   pipe_type   type;
   int         branch_num;
   double      length;
   double      diameter;
   double      friction_factor;
   sptr_pipe_t pipe_station;
   std::string name;
   int         i_sfrom;
   int         i_sto;
};

enum class pipe_type = { pipe, resistor, compressor, regulator,  valve};
```

Network graphs are typically populated using network data files provided by the user. We refer the reader to Section 5 for the structure of these data files and Section 5.5. However, for the sake of completeness in

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

presenting the numerical stage, it is also shown how to manually introduce pipes and nodes properties in the graph. This approach can be found in the unitary tests (shimmer++/unit_tests) designated to assess different modules of the code separately. We emphasize once again that using the network data file is the recommended approach. Nevertheless, manual input could be useful from the developer's perspective. Let's take as example the graph in Fig. 6 of a simplified gas network.

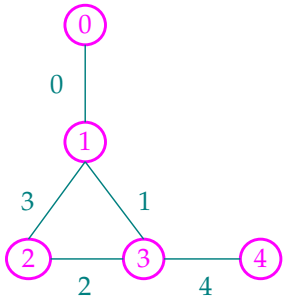


Figure 6: Graph representation of a simplified gas network.

6.1.1 Define the graph

In the main function add the definition of the graph. For the sake of clearness, let us proceed step by step and introduce two functions aimed to add first the vertex, `make_init_vertex` and later on the pipes properties, `make_init_pipes`.

```
int main()
{
    infrastructure_graph igraph;
    std::vector<vertex_descriptor> vds;
    make_init_vertex(igraph, vds);
    make_init_pipes(igraph, vds);

    return 0;
}
```

6.1.2 Add nodes specification

Let us suppose that the data shown in Tab. 7 is attached to the network vertices. For this example, no specification about mixture composition is taken into account. Nonetheless, this functionality is supported and can be easily added.

		Node properties			Mixture composition		
		G[kg/s]	p[Pa]	H[m]	CH4	N2	...
Nodes	0	5000	-60	10.0	-	-	-
	1	0	20	20.0	-	-	-
	2	0	25	30.0	-	-	-
	3	0	35	40.0	-	-	-
	4	0	50	50.0	-	-	-

Table 7: Parameters concerning node specification.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

In the following snippet code, you will distinguish a std::vector vds and a lambda function called add_vertex (see line 6), which internally calls the boost::add_vertex. The first one is used for the storage of all nodes created at the add_vertex call and it will be used in the following for the definition of the edges. For the sake of simplicity, the lambda add_vertex will not be further explained, but keep in mind it is needed for the management of internal pointers in the vertex properties, associated to its non-pipe elements.

```
1 static void
2 make_init_vertex(infrastructure_graph& igraph, std::vector<vertex_descriptor>& vds)
3 {
4     auto add_vertex = [&](vertex_properties&& vp)
5     {
6         auto v = boost::add_vertex(igraph);
7         igraph[v] = std::move(vp);
8         return v;
9     };
10
11     // Insert station config (name, no., pressure, flux, height)
12     vds.push_back( add_vertex( vertex_properties( "station 0", 0, 5000.,-60, 10. )) );
13     vds.push_back( add_vertex( vertex_properties( "station 1", 1,    0., 20, 20. )) );
14     vds.push_back( add_vertex( vertex_properties( "station 2", 2,    0., 25, 30. )) );
15     vds.push_back( add_vertex( vertex_properties( "station 3", 3,    0., 35, 40. )) );
16     vds.push_back( add_vertex( vertex_properties( "station 4", 4,    0., 50, 50. )) );
17 }
```

6.1.3 Add pipes specification

Let us now continue with the pipes specification using the values in Tab. 8.

		Nodes		Pipe properties		
		In	Out	L[m]	D[m]	epsi[m]
Pipes	0	0	1	80.0	0.6	1.2e-5
	1	1	3	90.0	0.5	1.3e-5
	2	3	2	100.0	0.4	1.4e-5
	3	1	2	110.0	0.3	1.5e-5
	4	3	4	120.0	0.2	1.6e-5

Table 8: Parameters concerning pipe specification.

In order to add the properties, first create an object of type edge_properties instantiating it with the data related to the pipes: length, diameter and factor epsi. Afterwards, the edges of the graph are created using the boost::add_edges function while insertions of the pipes properties are given as parameters. The first two entrances (arguments) are the nodes defining the edge (two first columns in Tab. 8) followed by the object with its corresponding properties.

```
1 static void
2 make_init_pipes(infrastructure_graph& igraph, std::vector<vertex_descriptor>& vds)
3 {
4     edge_properties ep0  = {pipe_type::pipe, 0,   80, 0.6, 1.2e-5};
5     edge_properties ep1  = {pipe_type::pipe, 1,   90, 0.5, 1.3e-5};
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

```
    edge_properties ep2  = {pipe_type::pipe, 2,  100, 0.4, 1.4e-5};
    edge_properties ep3  = {pipe_type::pipe, 3,  110, 0.3, 1.5e-5};
    edge_properties ep4  = {pipe_type::pipe, 4,   80, 0.2, 1.6e-5};

    boost::add_edge( vds[0], vds[ 1], ep0, igraph);
    boost::add_edge( vds[1], vds[ 3], ep1, igraph);
    boost::add_edge( vds[3], vds[ 2], ep2, igraph);
    boost::add_edge( vds[1], vds[ 2], ep3, igraph);
    boost::add_edge( vds[3], vds[ 4], ep4, igraph);
}
```

6.1.4 Incidence matrix A

The incidence matrix, as well as the graph, carries information about the connection between nodes and pipes. Hence, it is directly built from graph and used in the linear system on Section 6.3. See also Fig. 6. It is also used to exploit vectorization in some parts of the code, instead of calling the graph. The complete implementation is written in `Incidence.h`.

		Pipes				
		0	1	2	3	4
Nodes	0	1				
	1	-1	1		1	
	2			-1	-1	
	3		-1	1		1
	4					-1

Table 9: Incidence matrix based on the graph in Fig. 6.

Hands-on: Recall first to create a graph with the specification, as proposed in the previous examples. For this case, an slightly modification is done to summarize all in a unique function `make_init_graph`.

```
int main()
{
    // Define graph
    infrastructure_graph graph;
    make_init_graph(graph);

    // Build the incidence matrix A
    incidence inc(graph);

    // Get and print (std::cout) incidene matrix
    std::cout << "Incidence A: \n" << inc.matrix() << std::endl;
    std::cout << "Incidence A (only inlet): \n" << inc.matrix_in()<< std::endl;
    std::cout << "Incidence A (only outlet): \n" << inc.matrix_out()<< std::endl;

    return 0;
}
```

6.2 Stations

The nodes referred to as *"stations"* are those ones on which a external boundary condition is imposed. They rely on constraints that regulate either flux or pressure, depending on the nature of the station. A detailed

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

description of the possible nodal station types is given in 4.1. The list there provided is not exhaustive and developers can in principle add further stations with their control modes. As shown in Figure 6.2, stations can be of the exit/entry type with respect to the flux L, while junctions (not defined as stations) impose the sole requirement that $L = 0$ which is not exogenous but embedded in their definition as *"junction"*.

In what follows, we denote the control mode of a station as a boundary condition that imposes either pressure or flux with values specified by the user or fixed according to the type of station. Additional constraints regarding design parameters, user-defined set-points or user-defined parameters are also possible. The design parameters and the user-defined set-points are denoted henceforth as *"hard constraints"*, since a violation leads to a change of the control mode. In contrast, the user-parameters are denoted as *"soft constraints"*, since a violation triggers only a warning. These might represent the desired operating ranges thus the warning message returns an message about this "non critical" violation.

Nodal station which are able to change its control mode upon a violation of an hard constraint are modelled as multi-state stations.

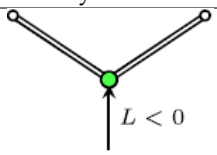
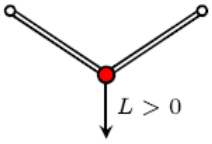
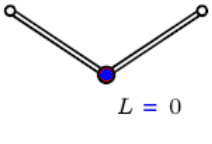
Entry Stations	Exit Stations	Junctions
		
$L < 0$	$L > 0$	$L = 0$
Control by pressure or inflow	Control by outflow	

Table 10: Classification of nodes by the direction of the external mass flow rate

Let us use a hypothetical gate station, whose operation is shown schematically in Figure 7, to aid in the exposition of the representation of a general station. The three boxes represent different types of behaviour, mainly defined by the control mode or boundary condition (BC), along with additional constraints specific to each case. Each of this behaviours is hereafter referred to as a state. Transitions from one state to another occur when hard constraints are violated. Each state encapsulates information about the condition imposed according to the control type, as well as the associated operational constraints and limits.

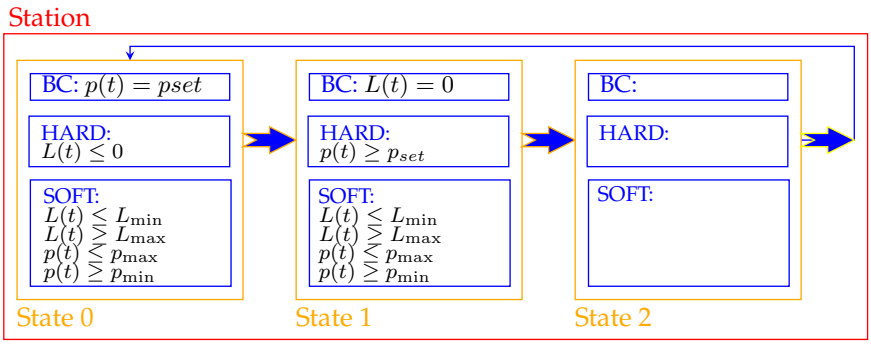


Figure 7: Shematic operation of a hypothetical 3-state station.

In shimmer++, we defined a station as a collection of states by using a vector container, that allows to add as many states are needed to capture the behaviour of the station.

```
class station
{
    std::vector<state> states_;

    void set_state(const state& s);
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

```
};
```

Each state is defined as a collection of constraints.

```
struct state
{
    constraint  boundary;
    constraint  hards;
    std::vector<constraint> softs;
};
```

```
class constraint
{
    hardness_type     hardness_;
    constraint_type   type_;
    vector_t          values_;
};
```

In order to add a new kind of station, a function must be created in the file boundary.h using either an object of type `one_state_station` or `multiple_states_station`.

6.2.1 One state station

For the sake of simplicity, let us add as an example the `consumption_wo_press`. Since there are no switch states, this station is defined as a one state station object. Inside the function only a state is defined, which implies a boundary, a hard constraint and the soft constraints. The function `build_user_constraints` defines the state components for each of the user-defined constraints, in an analogous manner to what was done for boundary and hard constraint types. The only difference is that `hardness_type::SOFT` must be used and multiple constraints are allowed.

```
template<typename VALUE>
one_state_station
make_consumption_wo_press(const VALUE& vals,
                          const std::vector<pair_input_t>&user_limits)
{
    // Define the state components
    auto s0_bnd = constraint(hardness_type::BOUNDARY,  // Set as boundary
                             constraint_type::L_EQUAL, // Only L_EQUAL or P_EQUAL
                             vals);                    // Boundary condition values

    auto s0_int = constraint(hardness_type::HARD,      // Set as hard constraints
                             constraint_type::L_GREATER_EQUAL, // Inequalities
                             0.0);                     // Limit of the inequality

    auto s0_ext = build_user_constraints(user_limits); // Create soft constraints

    // Create the state
    auto s0 = state(s0_bnd, s0_int, s0_ext);

    // Define the object consumption_station and give it a name
    one_state_station    consumption_station("CONSUMPTION_WO_PRESSURE");
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

```
    // Add the state to the station
    consumption_station.set_state(s0);

    return consumption_station;
}
```

6.2.2 Multiple states station

For station with more than one state, you should use `multiple_state_station` class. You are able to define stations with more than 2 states also. Just use the function set_state to add as many states as you please.

For example a two sate station:

```
    multiple_states_station remi("REMI_WO_BACKFLOW");
    remi.set_state(s0);
    remi.set_state(s1);
```

In the case of an hypothetical 3 states station, the quantity of states must be provided.

```
    multiple_states_station    hypo("HYPOTHETICAL STATION", 3);
    hypo.set_state(s0);
    hypo.set_state(s1);
    hypo.set_state(s2);
```

The complete definition of the station is done as shown for the `one_state_station`

```
template<typename VALUE_TYPE>
multiple_states_station
make_remi_wo_backflow(  const VALUE_TYPE& Pset,
                        const std::vector<pair_input_t>& user_limits_s0,
                        const std::vector<pair_input_t>& user_limits_s1)
{
  using hard_t   = hardness_type;
  using constr_t = constraint_type;

   // 1. STATES
   // 1.1 Define the first state
  auto s0_bnd = constraint(hard_t::BOUNDARY, constr_t::P_EQUAL, Pset);
  auto s0_int = constraint(hard_t::HARD,     constr_t::L_LOWER_EQUAL, 0.0);
  auto s0_ext = build_user_constraints(user_limits_s0);

  // 1.2 Define the second state
  auto s1_bnd = constraint(hard_t::BOUNDARY, constr_t::L_EQUAL, 0.0);
  auto s1_int = constraint(hard_t::HARD,     constr_t::P_GREATER_EQUAL, Pset);
  auto s1_ext = build_user_constraints(user_limits_s1);

  // 1.3 Create states
  auto s0 = state(s0_bnd, s0_int, s0_ext);
  auto s1 = state(s1_bnd, s1_int, s1_ext);
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

```
  // 2. STATION
  // 2.1 Create station
  multiple_states_station remi("REMI_WO_BACKFLOW");

  // 2.2 Add states
  remi.set_state(s0);
  remi.set_state(s1);

  return remi;
}
```

6.3 Numerical methods stage

The starting point for modelling gas fluid dynamics is the Navier-Stokes equations, along with some common model simplifications used in gas pipeline networks analysis. Here, we draw only the more general key points needed for the presentation of the tool. Let (v, p) be the velocity and pressure, then the standard model describing a fluid flow through a gas network is given by

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \dfrac{\partial (v\rho)}{\partial x} = 0, \\ \dfrac{\partial \rho v}{\partial t} + \dfrac{\partial}{\partial x}\left(\rho v^2 + p\right) + \dfrac{\lambda}{2D}\rho v|v| + \rho g \sin\alpha = 0. \end{cases} \tag{3}$$

where D denotes the diameter and α the inclination angle of the pipe. The first assumption used is unidimensional (1-D) flow through the pipes, which implies that radial variations are neglected. Another common simplification concerns gas transport and distribution, which it is typically supposed to occur within an isothermal process. This assumption holds in general due to the stability of the soil temperature where pipelines are normally installed. Situations in which the isothermicity assumpion is weaker are cases of open-air pipelines which are exposed to local weather, the proximity of an outlet section of a compression or a gas regulation station and underwater gasducts. However, overall this assumption seems to be a trade-off which allows avoiding to solve the energy equation leaving the problem of gas flow simulation as a pressure-velocity coupled problem, based on the coupled solution of the continuity equation (mass conservation equation) and of the momentum equation. The second main assumption is the creeping motion one, which derives from the low gas velocity within the pipeline with respect to the gas speed of sound (generally: $25m/s$ vs $\approx 400m/s$). This allows to neglect the convective term. Some estimations of pressure drop error based on these assumptions are discussed in [3]. A schematic resume of the modelling is shown in Fig. 8. The assumption above applied on the flow equation, that is applied to a pipeline (or a pipeline section) whose length is Δx. The continuity equation is instead applied to each node of the network, where the mass balance is performed over the volume V_i. A complete derivation of the modelling system can be found in [3].

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

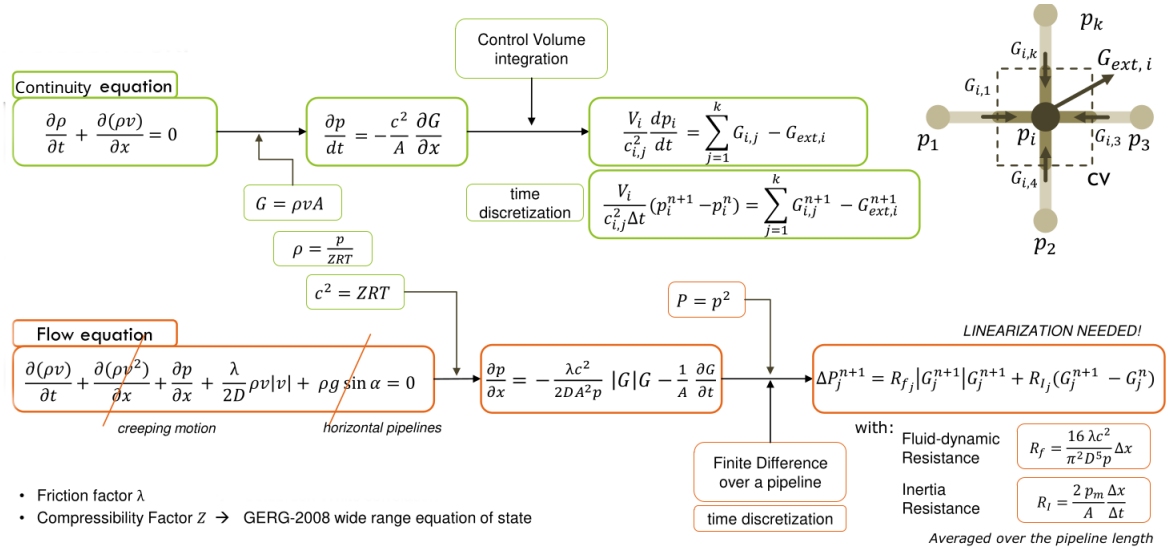


Figure 8: Schematic transient gas network model.

The resulting variables after linearization and discretization are pressure $\mathbf{p}$ and mass flow rate $\mathbf{L}_{rate}$ (G_{ext} in the scheme) attached to the node volumes, and flux $\mathbf{G}$ attached to pipe volumes. In Shimmer++, these set of variables are all gathered in an structure named "variable" as follows

```
struct variable
{
    vector_t pressure;
    vector_t flux;
    vector_t L_rate;

    // Constructors
    variable();
    variable(const vector_t&p, const vector_t&f ,const vector_t&l);
};
```

The structure of the methodology for the numerical methods stage can be seen as an outer loop that handles the unsteady (transient) time advancement, an inner loop solving the gas fluid dynamics, and a step dedicated to the quality tracking (see also Fig. 2) .

In Fig.8, the friction-losses term in the flow equation (the fourth term from the left) has been modeled through the Darcy-Weisbach equation, as it is evident form the expression. This equation express the pressure losses due to the pipe-wall friction as a function of the fluid density, the squared of its velocity, some geometrical parameters and a so-called *friction factor* λ. This term is in general a function of the pipeline inner wall roughness ε_i, the inner diameter D_i, and the Reynolds Number Re, denoting the regime of the fluid flow. The Reynolds number is in turn a function of the fluid density, its velocity, the pipe diameter and the dynamic viscosity μ.

Consequently, besides the coupled mass-momentum equations, further relations are needed, specifically

- the friction factor correlation
- the viscosity correlation for a gas-mixture
- the Equation of State for the determination of the compressibility factor.

As for the latter, the need for an Equation of States emerges to set a correlation between gas pressure and density, as it is schematically displayed in Fig.8.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

Remark. *The gas network transport solver, hereafter referred as the time solver and defined in Section 6.5, is designed to take as C++ template parameters the equation of state and the viscosity, thereby enabling the use of different formulations for each. The friction factor might adopt a similar template-approach, instead of a solely function. Future developments may easily extend this methodology to achieve a more flexible and consistent framework.*

In the next sections, these correlation are briefly illustrated, enabling the user/developer to possibly modify them.

6.3.1 Friction factor

In the literature, there is a wide range of mathematical models to describe the friction factor. Shimmer++ relies on the Cheng formula correlation [2], written as follows

$$\frac{1}{\lambda} = \left(\frac{Re}{64}\right)^{\alpha} \left[1.8 \log\left(\frac{Re}{6.8}\right)\right]^{2(1-\alpha)\beta} \left[2.0 \log\left(\frac{3.7D}{\varepsilon}\right)\right]^{2(1-\alpha)(1-\beta)},$$

with shorthand notation:

$$\alpha = \left(1 + \left(\frac{Re}{2720}\right)^{9}\right)^{-1}, \qquad \beta = \left(1 + \left(\frac{Re\,\varepsilon}{160D}\right)^{2}\right)^{-1}.$$

where Re is the Reynolds number, D the inner diameter and of the pipe and ε is the internal roughness of the pipe-wall.

Extensions to other models (e.g. HOFER, Colebrook-White etc.) can be easily added by substitution of the implementation of the Cheng formula shown here below.
See the pipe configuration file `shimmer/shimmer++/src/solver/pipe_calculator.cpp`.

```
double
friction_factor_average(const edge_properties& pipe, const double & Temperature,
                        const double & flux, const double & mu)
{
    auto Re = std::abs(flux) * pipe.diameter / (pipe.area() * mu) ;

    auto eps_over_d = pipe.friction_factor / pipe.diameter;
    auto a = 1.0 / (1.0 + std::pow( Re  / 2720.0, 9));
    auto b = 1.0 / (1.0 +  std::pow(Re * eps_over_d/160.0, 2.0) );

    auto t0 = 3.7 / eps_over_d;
    auto t1 = std::pow( 64.0 / Re, a) ;
    auto t2 = std::pow( 1.8 * std::log10(Re/6.8),  2.0 * (a -1.0) * b);
    auto t3 = std::pow( 2.0 * std::log10(t0), 2.0 * (a - 1.0) * (1.0 - b));

    return t1 * t2 * t3;
}
```

6.3.2 Viscosity

At the present time, only a viscosity model accounting for complex composition of gases is supported. It is here labeled in honor to one of the authors, Kukurugya. The alternative for more simplified problems is the constant viscosity model, that cannot be modified at running time and it is set accordingly as $\mu = 1e-5\ [Pa \cdot s]$.
See the pipe configuration file `shimmer/shimmer++/src/solver/viscosity.cpp`.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

```
enum viscosity_type
{
    Kukurugya,
    Constant,
};
```

Hands-on: The type of viscosity is given as a template parameter to the viscosity function as shown hereafter. Notice that the viscosity takes the same temperature for each edge in the graph. This is in agreement with the isothermal hypothesis of the gas network. However, straightforward modifications can be done if variations in space for temperature are desired.

```
int main()
{
    // Average temperature
    double Tm = 293.15; // [K]

    // Define graph
    infrastructure_graph graph;
    make_init_graph(graph);

    // Compute viscosity
    std::cout << "mu_k: \n" << viscosity<viscosity_type::Kukurugya>(Tm, graph);
    std::cout << "mu_c: \n" << viscosity<viscosity_type::Constant>(Tm, graph);

    return 0;
}
```

6.3.3 Equation of state - Gas dynamics relations

Industry have been using modifications to ideal gas state to accommodate to their applications and constraints. A compilation of this bunch of equations used by industrial operators was gathered in [5]. Some common formulas are named under acronyms to identify them, from where the most common ones are PAPAY, AGA-8, sGERG-88 and GERG-2008. In this work, we use the PAPAY formula and the GERG-2008 equation.

The PAPAY formula [6] indeed is a correlation for the compressibility factor Z known to be valid for pressure up to 150 $[bar_g]$. The formulation is as follows

$$Z = 1 - 3.52\left(\frac{p}{p_c}\right)\exp\left[-2.260\left(\frac{T}{T_c}\right)\right] + 0.274\left(\frac{p}{p_c}\right)^2\exp\left[-1.878\left(\frac{T}{T_c}\right)\right],$$

where p_c and T_c are the critical pressure and critical temperature of the mono-component gas, respectively. Extensions of the PAPAY correlation to multicomponent gas mixtures are possible by using the pseudo-critical pressures and temperatures for the gas mixtures. In Shimmer++, only the equation for methane (CH_4) is provided, as other complex gases can be modelled using the GERG-2008.

The GERG-2008 equation is instead the correct equation to be adopted in case of multi-component based natural gas modelling. It is a relative recent equation of state written in terms of the Helmholtz free energy. It is used as an ISO standard and it is known to be highly flexible with the number of components considered for the mixture. The full description of the equation is given in [1]. The standard implementation is provided by the AGA8 repository maintained by NIST and has been adapted for our purposes.

To review the available equations of state or to include a new one, refer to the project file `shimmer/shimmer++/src/solver/gas_law.h`.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

Hands-on: To include a new equation of state, a derived class inheriting from the base class `equation_of_state`, must be implemented. Here below, we show only some guidelines of the base class, to see the complete/accurate implementation we refer the reader to the file `shimmer/shimmer++/src/solver/gas_law.h`.

```
class equation_of_state
{
    virtual void initialization(linearized_fluid_solver *) = 0;
    virtual void mixture_molar_mass(const matrix_t&, const matrix_t&) = 0;
    virtual auto molfrac_2_massfrac(const infrastructure_graph&, const incidence&) = 0;
    virtual auto massfrac_2_molfrac(const matrix_t&, const matrix_t&) = 0;
    virtual auto speed_of_sound(linearized_fluid_solver *) const = 0;
};
```

Example of the derived class for the PAPAY equation of state

```
class papay: public equation_of_state
{
public:
    papay();

    void initialization(linearized_fluid_solver *lfs);
    void mixture_molar_mass(const matrix_t&, const matrix_t&);
    auto molfrac_2_massfrac(const infrastructure_graph&, const incidence&);
    auto massfrac_2_molfrac(const matrix_t&, const matrix_t&);
    vector_t compute_Z(double temperature, const vector_t& pressure) const;

    auto speed_of_sound(linearized_fluid_solver *lfs) const;
};
```

6.4 Fluid solver

We denote as fluid solver the innermost iteration loop where the Navier-Stokes equations are solved following the ideas presented in [3], Section 4 and schematized in Fig.8. The resulting pair of discretized equations are shown in Fig. 9.

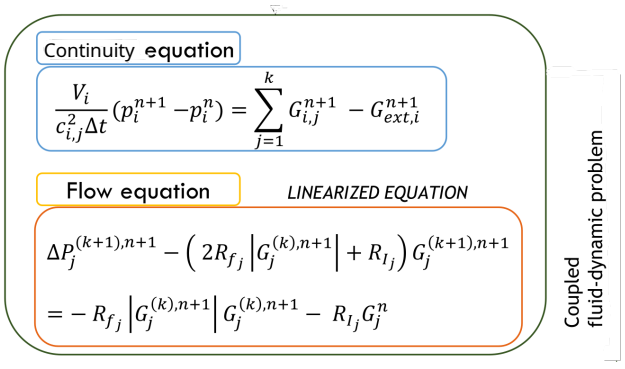


Figure 9: Discretized mass and momentum equations.

The linear system resulting after linearization and discretization of the Navier-Stokes equations can be represented as in Fig.10.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

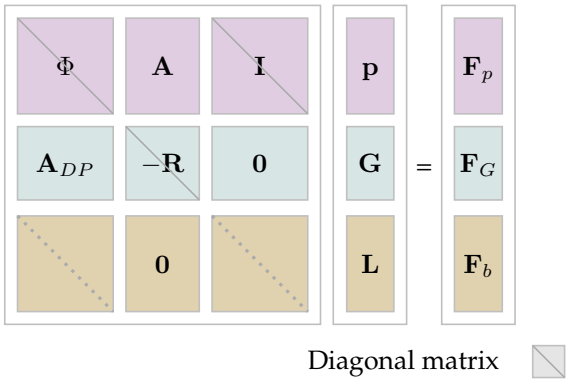


Figure 10: Algebraic system after linearization of the fluid dynamic equations.

where **I** stands for the identity matrix and recall that **A** stands for the incidence matrix, **R** stands for the resistance matrix which incorporates both the fluid-dynamic resistance R_f and the inertia resistance R_I, which is relevant for unsteady (transient) simulation problems, while it is zero when the steady-state simulation is addressed. See Fig.8 for the specific definition of the two terms for the resistance and [3] for the step-by-step matrices construction. The code implementation search to mimic the algebraic system in Fig. 10.

```
void linearized_fluid_solver::run(...)
{
    var_.pressure = var_guess.pressure;
    var_.flux   = var_guess.flux;
    var_.L_rate = vector_t::Zero(num_nodes_);

    eos->initialization(this);

    for(size_t iter=0;iter<=MAX_ITERS;iter++)
    {
        press_pipes_ = average(var_.pressure, inc_);
        auto [c2_nodes, c2_pipes] = eos->speed_of_sound(this);

        auto mass_system = continuity( ...);
        auto mom_system  = momentum( ...);
        auto bnd_system  = boundary(...);

        auto [LHS, rhs]= assemble(mass_system, mom_system, bnd_system);

        Eigen::SparseLU<sparse_matrix_t> solver;
        solver.compute(LHS);
        vector_t sol = solver.solve(rhs);

        if (convergence(sol))
           return;
    }
}
```

The fluid solver accommodates steady and unsteady runs, depending on the setting of the input parameters. The object is constructed using

```
linearized_fluid_solver lfs(n, unsteady, tol, dt, temperature, mu, inc, graph);
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

The input parameters are listed here below

- Integer `n`: denotes the time interval t^n. For steady runs $n = 0$.
- timestep Δt: only needed in case of transient runs.
- unsteady: `false` if steady state, `true` in unsteady state.
- tolerance: set the tolerance for the convergence of the iterative solver
- temperature: constant network temperature
- viscosity `mu`: see Section 6.3.2
- incidence `inc`: see Section 6.1.4
- graph: see Section 6.1

To find the variables for the fluid flow in an steady state, make `timestep=0` and `unsteady=false`. The iterative solver is performed by the fluid solver run function

```
lfs.run(area_pipes, var0, var, &eos);
```

It requires an initial object `variable var0`, that acts either as an initial condition in the unsteady or as a guess variable in the steady state.

Hands-on: For the complete example check `unit_tests/test_fluid_solver.cpp`.

```
int main()
{
    // 1. Create the arguments needed for the init of the linearized fluid solver
    // 1.1 Numerical and physical setting
    bool unsteady = true;
    double temperature = 293.15; // [K]
    double dt  = 180.0; // [s]
    double tol = 1.E-04;

    // 1.2 Graph and derived data
    infrastructure_graph graph;
    make_init_graph(graph);
    incidence inc(graph);
    auto mu = viscosity<viscosity_type::Kukurugya>(temperature, graph);

    // 2. Create the arguments needed to run the fluid solver
    vector_t area_pipes = area(graph);

    gerg_aga gerg_eos;
    gerg_eos.mixture_molar_mass(graph, inc);

    // 3. Fluid dynamics solver
    linearized_fluid_solver lfs(0, unsteady, tol, dt, temperature, mu, inc, graph);
    lfs.run(area_pipes, var, var, &gerg_eos);

    return 0;
}
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

6.5 Time solver

The advance in time is implemented in the time solver, see *time_solver.hpp*, which allows for the steady and unsteady (transient) state. It corresponds to the outer loop of the numerical methods stage. At the moment of the creation of this time solver object, the viscosity model along with the equation of state must be provided as template parameter, i.e. at compile time. At the moment of the writing of this document, only two models are available GERG-2008 and PAPAY [6], the latter for pure methane only. Thus, assuming constant viscosity model, the two available options are as follows:

```
time_solver<gerg_aga,  viscosity_type::Constant> ts(graph, ...);
```

or

```
time_solver<papay, viscosity_type::Constant> ts(graph, ...);
```

To initialize the object, physical and numerical properties that remain unchanged during the simulation must be provided, as the temperature of the network which is assumed to be constant. The time solver is divided into: the initialization and the advancement steps. The first aims at solving a physical initial condition by solving a steady fluid-dynamic iteration, provided a guess solution and a tolerance (`tol_std`) are given. The second step is devoted to evolution in time and therefore the number of steps `num_steps`, the timestep `dt` and the tolerance `tol` for the transient fluid-dynamic iterations must be provided.

Hands-on: For the complete example check `unit_tests/test_time_solver.cpp`.

```cpp
int main()
{
    infrastructure_graph graph;
    make_init_graph(graph);

    //Define the type o time solver: eq of state and viscosity model
    using time_solver_t = time_solver<papay, viscosity_type::Constant>;
    // Initialize time solver object
    time_solver_t ts(graph, temperature);
    // Run simulation with steady state
    ts.initialization(guess_std, dt_std, tol_std);
    // Run simulation with unsteady state
    ts.advance(dt, num_steps, tol);

    return 0;
}
```

6.6 Quality tracking

This section is devoted to the description of the model and the numerical discretization of the flow of a mixture of various gaseous species, notably the quality tracking. A Lagrangian coordinates approach was proposed in [7]. Despite its novelty and its ability to capture discontinuities, the method is prone to cumulative errors, as is pointed out in [8]. Let us take as an example the simple gas network shown in Figure 11, which illustrates non uniform refinement along the network pipelines.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

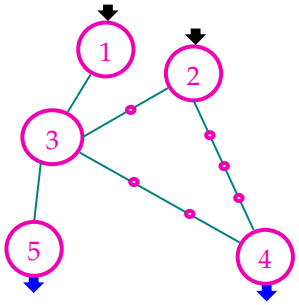


Figure 11: Example of a gas network. Color scheme: magenta (nodes) and teal (pipes). Circle with numbers denote the stations and dots represent fictitious stations due to the refinement.

We consider a gas mixture composed of N_α ideal gas constituents. Each constituent is modeled as an ideal compressible gas. In addition, they are assumed to be inviscid and chemically inert.

Remark. *Vaccum is not allowed in any part of the pipeline.*

We additionally assumed thermal equilibrium. Let ρ and p denote the density and pressure of the mixture, respectively, with the later computed from the equation of state. Let v denote the one-dimensional velocity of the mixture in a pipe and also the individual velocity of each species, as the effects of molecular diffusion are neglected [9]. Let ρ_α denote the individual material density and the partial density d_α of the α-th gas component. The latter is defined as an additive partial density such the global mixture density can be expressed as follows

$$\rho = \sum_{\alpha=1}^{N_\alpha} d_\alpha \qquad \text{and} \qquad 1 = \sum_{i=1}^{N_\alpha} \frac{d_\alpha}{\rho}.$$

Here, the second assertion is just a direct consequence of the first. The partial density of the gas is related to the individual material density ρ_α, as $d_\alpha = \rho_\alpha \cdot \gamma_\alpha$, where $\gamma_\alpha := V_\alpha/V$ denotes the volumetric fraction acting here as correction factor.

6.6.1 The model for transport of gas species

Henceforth, the governing equations we consider are the one-dimensional mass and momentum Euler equations, where chemical reactions sources are not taken into account since the gas components are chemically inert. A well-known presentation of the mass conservations is written in terms of the mass fraction quantities:

$$\begin{cases} \dfrac{\partial \rho Y_\alpha}{\partial t} + \dfrac{\partial}{\partial x}(\rho Y_\alpha v) = 0, \quad \alpha \in \{1, \cdots, N_\alpha\} \\ \dfrac{\partial \rho v}{\partial t} + \dfrac{\partial}{\partial x}(\rho v^2 + p) = 0. \end{cases} \tag{4}$$

In [9], it was also shown that the system (4) can be rewritten using the density additive property as follows

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \dfrac{\partial}{\partial x}(\rho v) = 0, \\ \dfrac{\partial \rho Y_\alpha}{\partial t} + \dfrac{\partial}{\partial x}(\rho Y_\alpha v) = 0, \quad \alpha \in \{1, \cdots, N_\alpha - 1\}, \\ \dfrac{\partial \rho v}{\partial t} + \dfrac{\partial}{\partial x}(\rho v^2 + p) = 0, \end{cases} \tag{5}$$

but also that the first and the second equations result in the non-conservative mass fraction equation

$$\frac{\partial Y_\alpha}{\partial t} + v\frac{\partial Y_\alpha}{\partial x} = 0, \qquad \alpha \in \{1, \cdots, N_\alpha - 1\}, \tag{6}$$

implying that mass fractions Y_α are being transported. This is found in the literature as passive transport, see [10].

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

6.6.2 Transport through pipes: quality tracking

Let us consider as model problem the one-dimensional conservation law for a quantity $q(x,t)$ with flux $F(q(x,t))$

$$\frac{\partial q}{\partial t} + \frac{\partial F(x,t)}{\partial x} = 0. \tag{7}$$

Let $\bigcup I_i$ be the partitioning of a generic pipe domain L, where $I_i =]x_{i-1/2}, x_{i+1/2}[$ denotes the interval with size $\Delta x_i = x_{i+1/2} - x_{i-1/2}$ and barycenter $x_i = (x_{i+1/2} + x_{i-1/2})/2$. The time horizon is denoted by $T > 0$ Let

$$Q_i^n = \frac{1}{\Delta x_i} \int_{I_i} q(x, t^n) dx.$$

using the Lax-Wendroff second order scheme to solve hyperbolic equations, the general discretized form is

$$\frac{Q_i^{n+1} - Q_i^n}{\Delta t} + \frac{F_{i+1/2}^{n+1/2} - F_{i-1/2}^{n+1/2}}{\Delta x} = 0. \tag{8}$$

The Lax-Wendroff method was introduced in the seminal work [11] in 1960 for hyperbolic conservation laws and named after the two authors. Upon defining the quantities

$$F_{i-1/2}^{n-1/2} = \frac{1}{2}\left(F_{i-1}^n + F_i^n\right) - \frac{1}{2}\frac{\Delta t}{\Delta x}\mathcal{A}_{i-1/2}^n\left(F_i^n - F_{i-1}^n\right), F_{i+1/2}^{n+1/2} = \frac{1}{2}\left(F_{i+1}^n + F_i^n\right) - \frac{1}{2}\frac{\Delta t}{\Delta x}\mathcal{A}_{i+1/2}^n\left(F_{i+1}^n - F_i^n\right).$$

the Lax-Wendroff method can be implemented in the standard one step formulation which requires the computation of $\mathcal{A} = \partial_q f(q(x,t))$.

The Lax-Wendroff method is derived for conservative equations. Thus, we do not use the non-conservative transport equation (6) and instead we solve the system in a decoupled fashion (5). Let $r_i = \frac{Q_i - Q_{i-1}}{Q_{i+1} - Q_i}$, then we use the flux limiter $\Phi(r)$ named Superbee and defined as follows

$$\Phi(r) = \max(0,\ \min(1, 2r),\ \min(2, r)).$$

6.6.3 Nodal mass balance: Admixing

For the quality tracking, we assume nodes with volume. Thus, accumulation of mass, momentum or energy can happen. We also assume perfect mixing, so with all input flows a homogeneous mixture is created and it is distributed proportionally to all output pipes. Let $\Omega^\bullet$ be a node domain with boundary $\Gamma^\bullet$, which can be further seen as $\Gamma^\bullet = \Gamma_+^\bullet \cup \Gamma_-^\bullet$, being the cross section inlet and outlet boundaries of the pipe, respectively.

$$\int_{\Omega^\bullet} \frac{\partial \rho Y_\alpha}{\partial t} + \int_{\Omega^\bullet} \operatorname{div}(\rho \mathbf{v} Y_\alpha) = 0, \tag{9}$$

The second term can be reformulated by the Divergence Theorem on the α-th mass fraction flux. As the boundary of a pipe domain can be seen has the collection of faces f, the previous equation can be easily reformulated as

$$\sum_{f \subset \Gamma^\bullet} (\rho \mathbf{v} Y_\alpha A)_{|f} \cdot \mathbf{n}_f = \sum_{f \subset \Gamma_+^\bullet} (\phi Y_\alpha A)_{|f} - \sum_{f \subset \Gamma_-^\bullet} (\phi Y_\alpha A)_f$$

with $\rho \mathbf{v} \cdot \mathbf{n} = \phi n$ with n unitary value positive and negative on $\Gamma_+^\bullet$ and $\Gamma_-^\bullet$, respectively.

$$\sum_{f \in \Gamma_+^\bullet} (\phi Y_\alpha A)_f = Y_{\alpha|f_{\bullet\leftarrow}} \dot{m}_{\bullet\leftarrow} + \sum_{f \subset \Gamma_+^\bullet \setminus \Gamma_{\bullet\leftarrow}} (\phi Y_\alpha A)_{|f},$$

$$\sum_{f \subset \Gamma_-^\bullet} (\phi Y_\alpha A)_{|f} = Y_{\alpha|f_{\bullet\rightarrow}} \dot{m}_{\bullet\rightarrow} + \sum_{f \subset \Gamma_-^\bullet \setminus \Gamma_{\bullet\rightarrow}} (\phi Y_\alpha A)_{|f},$$

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

Remark (Perfect mixing). *we assume perfect mixing at the node, thus all output faces, $f \subset \Gamma^\bullet_-$, the same mixture is provided. Whence, the mass fractions on the output faces are equal to the mass fractions on the node, i.e $Y_{\alpha|f} = Y_{\alpha,\bullet}$ for all $f \in \Gamma^\bullet_-$.*

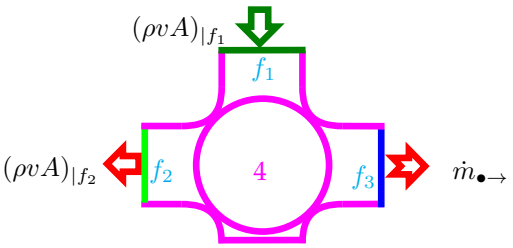


Figure 12: Perfect mixing: mass fractions on output faces f_2 and f_3 are equal.

Let us now address the temporal term

$$\int_{\Omega^\bullet} \frac{\partial \rho Y_\alpha}{\partial t} \approx \Omega^\bullet \frac{\rho^{n+1}_\bullet Y^{n+1}_{\alpha,\bullet} - \rho^n_\bullet Y^n_{\alpha,\bullet}}{\Delta t}.$$

Remark (Variation in time of c^2). *Using gas relations, the variation in time $\frac{\partial \rho}{\partial t}$ could be reformulated as $\partial_t \rho = \partial_x (p/c^2)$. Hence,*

$$\int_{\Omega^\bullet} \frac{\partial \rho Y_\alpha}{\partial t} \approx \frac{\Omega^\bullet}{\Delta t} \left(\left(\frac{p_\bullet Y_{\alpha,\bullet}}{c^2_\bullet} \right)^{n+1} - \left(\frac{p_\bullet Y_{\alpha,\bullet}}{c^2_\bullet} \right)^{n} \right).$$

Thus, putting all together

$$\left(\frac{\Omega^\bullet}{\Delta t} \rho^{n+1}_\bullet + \dot{m}_{\bullet\rightarrow} + \sum_{f \subset \Gamma^\bullet_- \setminus \Gamma_{\bullet\rightarrow}} (\phi A)_{|f} \right) Y^{n+1}_{\alpha,\bullet} = \frac{\Omega^\bullet}{\Delta t} \rho^n_\bullet Y^n_{\alpha,\bullet} + Y^n_{\alpha|f_{\bullet\leftarrow}} \dot{m}_{\bullet\leftarrow} + \sum_{f \subset \Gamma^\bullet_+ \setminus \Gamma_{\bullet\leftarrow}} (\phi Y_\alpha A)^n_{|f}.$$

6.6.4 Quality tracking solver

The solver for the quality tracking is similarly defined as the time solver shown in Section 6.5. Thus, an equation of state and a viscosity law (Section 6.3.2) must be chosen to define the quality tracking type.

```
using solver_t = shimmer::qt_solver<shimmer::gerg_aga,shimmer::viscosity_type::Constant>;
```

To create the quality tracking object, 4 arguments are needed:

- infrastructure with the whole information of the network
- temperature of the network
- a boolean specifying if pipes must be refined, default `false`.
- number of time steps to run an unsteady state

It is important to distinguish steady and transient runs, see Table 6.6.4. In the first one, only admixing is made on each node, independently of refinement settings. The steady solver iterates until convergence with each iteration made of a fluid-dynamic run followed by a computation of the mass fractions. In the second case, the steady case is only intended to find the fluid-dynamics variables on the network as an initial condition. The initial mass fraction state is set as only methane; therefore, during the transient run, the mixture of gasses injected into the network will transport and the fluid-dynamics will change accordingly.

```
solver_t qt(infrastructure, temperature, do_refine, num_steps);
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

Time regime	Refinement	Mass fraction transport
Steady	No	Admixing
	Yes	Admixing
Unsteady	No	Admixing
	Yes	Admixing + QT

Recall that `config.quality_tracking` must be set `true` and `config.qt_steady` is set to `false` by default. The variable `config.qt_steady` is only used in case `config.quality_tracking` = `true`.

See the unitary test for the quality tracking solver denoted as `/unit_tests/test_qt`.

```
//Define the type o time solver: eq of state and viscosity model
using solver_t = shimmer::qt_solver<shimmer::gerg_aga,
                                    shimmer::viscosity_type::Constant>;
// Initialize quality solver object
solver_t qt(infra, temperature, refine, num_steps);

if(steady)
{
    // Run simulation with only admixing
    qt.steady_state(guess, dt_std, tol_std);
}
else
{
    // Run Fluid dynamics steady state
    qt.initialization(guess, dt_std, tol_std);
    // Run unsteady state
    qt.advance(dt, tol);
}
```

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

7 Conclusions

This SHIMMER project deliverable highlights the outcome of the Work Package 4, Task 4.2.4 which consisted in the development of an open-source fluid-dynamic model with gas quality tracking, called Shimmer++.

The model is publicly accessible at the GitHub repository here: `https://github.com/shimmerhydrogen/shimmer` Before the finalization of this document, a Demonstration Workshop has been held on line on Nov. 26^{th} 2025 in presence of the research and industrial partners involved in the SHIMMER Project where the main contents of this deliverble-handbook have been presented and a live demonstration has been given. The presentation is among the supporting material within the GitHub repository. It has been successfully validated against a commercial software and data from the industrial partners of the Work Package 4, as per the programmed milestone of the project.

The Shimmer++ model has been selected as a Key Exploitable Results of the project to participate to the EU Booster Service program for the exploitation.

Now it is publicly available and the authors see promising possibilities to exploit this outcome already starting by the academic and research environment in which it has been developed, as it is a powerful tool to be used in training students, early researcher and professional to get acquinted to the simulation of gas networks under non-conventional gas injection. Furthermore, it can be use to advance the research activities on gas networks.

Its features of transparency and flexibility which have driven the coding desing are aimed to allow researchers and industrial stakeholders to expand further, improve and integrate the model with others, to contribute to the technological advancement in the field of gas infrastructures.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement n° 101111888.

References

[1] O. Kunz and W. Wagner, "The gerg-2008 wide-range equation of state for natural gases and other mixtures: An expansion of gerg-2004," *Journal of Chemical & Engineering Data*, vol. 57, no. 11, pp. 3032–3091, 2012.

[2] N.-S. Cheng, "Formulas for friction factor in transitional regimes," *Journal of Hydraulic Engineering*, vol. 134, no. 9, pp. 1357–1362, 2008.

[3] M. Cavana, *Gas network modelling for a multi-gas system*. PhD thesis, Politecnico di Torino, Sept. 2020. PhD thesis, 32nd cycle (2016–2019).

[4] T. A. Gunawan, M. Cavana, P. Leone, and R. F. Monaghan, "Solar hydrogen for high capacity, dispatchable, long-distance energy transmission – a case study for injection in the greenstream natural gas pipeline," *Energy Conversion and Management*, vol. 273, p. 116398, 2022.

[5] T. Guillot, *Equation of State*, pp. 743–744. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015.

[6] J. Pápay, "A termeléstechnológiai paraméterek változása a gáztelepek müvelése során," *OGIL Müszaki Tudományos Közlemények*, pp. 267–273, 1968.

[7] M. Chaczykowski, F. Sund, P. Zarodkiewicz, and S. M. Hope, "Gas composition tracking in transient pipeline flow," *Journal of Natural Gas Science and Engineering*, vol. 55, pp. 321–330, 2018.

[8] Z. Zhang, I. Saedi, S. Mhanna, K. Wu, and P. Mancarella, "Modelling of gas network transient flows with multiple hydrogen injections and gas composition tracking," *International Journal of Hydrogen Energy*, vol. 47, no. 4, pp. 2220–2233, 2022.

[9] B. Larrouturou and L. Fezoui, "On the equations of multi-component perfect or real-gas inviscid flow," in *Nonlinear Hyperbolic Problems* (C. Carasso, P. Charrier, B. Hanouzet, and J. Joly, eds.), vol. 1402 of *Lecture Notes in Mathematics*, pp. 69–98, Berlin, Heidelberg: Springer, 1989.

[10] F. Bouchut, *Nonlinear Stability of Finite Volume Methods for Hyperbolic Conservation Laws: and Well-Balanced Schemes for Sources*. Frontiers in Mathematics, Birkhäuser Basel, 2004.

[11] P. Lax and B. Wendroff, "Systems of conservation laws," *Communications on Pure and Applied Mathematics*, vol. 13, no. 2, pp. 217–237, 1960.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.

The research leading to these results has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation under grant agreement nº 101111888.