Future harmonization and access to ESA environment and effects models

Simon Clucas

29/05/2019

Purpose of this talk

I worry about model and data interoperability

- **Basis to solicit feedback**
- **Network of Models proof-of-concept to explore possible solutions**
- **Hopefully start to form the basis of some standards**

*** There are several activities pursuing some of these aspects ***

As of today…

- The space weather and effects community has produced a plethora of tools, services, databases and models.
- They are written in an eclectic mix of … ok mostly Fortran but also python, C, C++ … , to a wide range of standards.
- They all have unique input and output formats including namelists, CSV files, carrier Pidgeon and Morse code.
- Some require an IT degree to install, maintain or even access

Therefore …

- End users can spend significant time with IT and data issues and not science
- It can be impossible to use these models and data sources together

Is this true?

My problem … there is more and more requirements on these models to support future services

Current snapshot

Small, non–exhaustive list for example only

SPENVIS (https://www.spenvis.oma.be/)
hosted: yes (BIRA) | web: yes | api: no | extensible: involved

ESPREM
hosted: no | web: no | api: yes | extensible: less involved

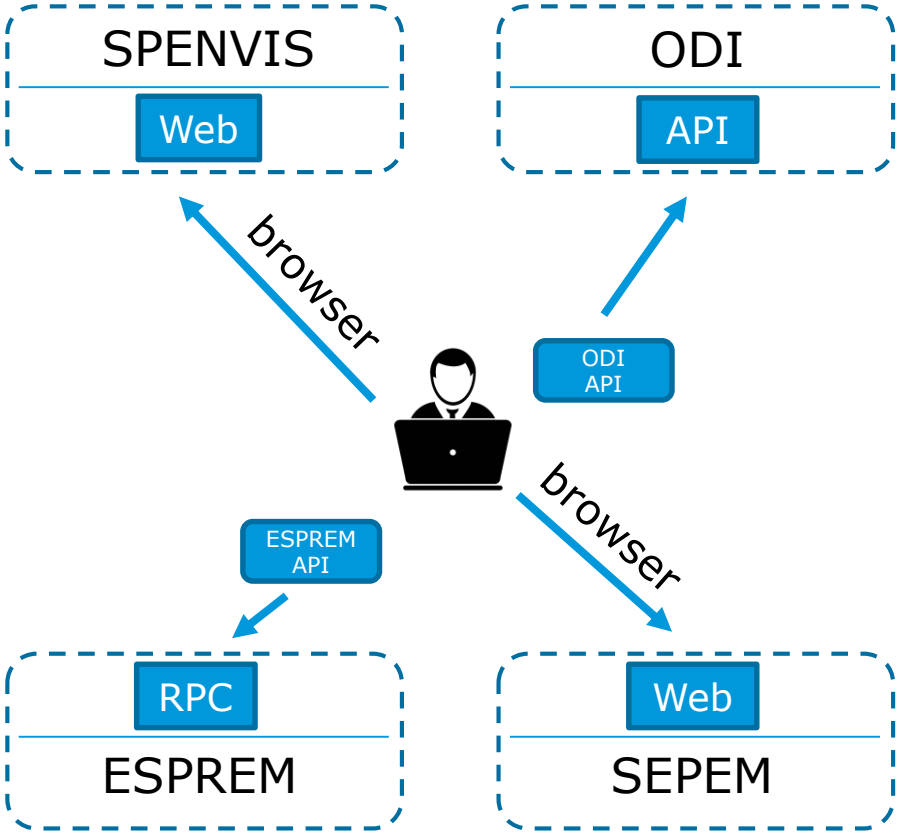
Open Data Interface (ODI)
hosted: yes (ESA) | web: no | api: yes (REST) | extensible: yes

Solar Energetic Particle Environment Modelling (SEPEM) (http://www.sepem.eu/)
hosted: yes (BIRA) | web: yes | api: no | extensible: no

Space Environment Database and Analysis Tools (SEDAT)
hosted: yes-ish (SSA) | web: yes | api: no | extensible: no

+ plus many more

Current state of affairs



1. Use ODI API to retrieve particle spectra
2. Try to find out what format SPENVIS needs
3. Process particle spectra data and convert to a form that can be uploaded to SPENVIS
4. Upload to SPENVIS and run model using web browser
5. Copy/paste results from SPENVIS web output
6. Write script to process SPENVIS results and plot
7. Do something amazing with ESPREM
8. Copy/paste results and write another a script to process ESPREM data
9. Plot ESPREM results along with SPENVIS data
10. Repeat steps 1-9 when you find out you used the wrong particle data
11. Come back in a year and get a colleague to reproduce your analysis

** One or more of these steps in not even possible! **

Current state of affairs

Summary of issues

- **Model discoverability**
- **Model documentation can be out of date**
- **Models and data sources cannot easily be pipelined into a workflow**
- **Data analysis involving iteration/loops can be impossible**
- **Including (or swapping in) newer models into a workflow normally involves a lot of work**
- **Reproducibility can be tough especially if significant time has elapsed and models have been 'upgraded'**
- **Reporting can be error prone due to missing or incorrect meta data in model outputs**

Any more?

Network of Models (NoM) Concept

Goals

Single client API to run all NoM models

- Easier discovery of models and data
- Less custom scripting and validating of data

Lightweight, general facade to convert any model into a **NoM server**

- Can retro-fit existing projects
- Simple interface spec for new projects
- More time on model development
- Less requirement to create custom API

Standardise on model description:

- Model docs
- Model meta data (version, provider etc.)
- Input provision format
- Output formats
- Dynamic model GUI creation

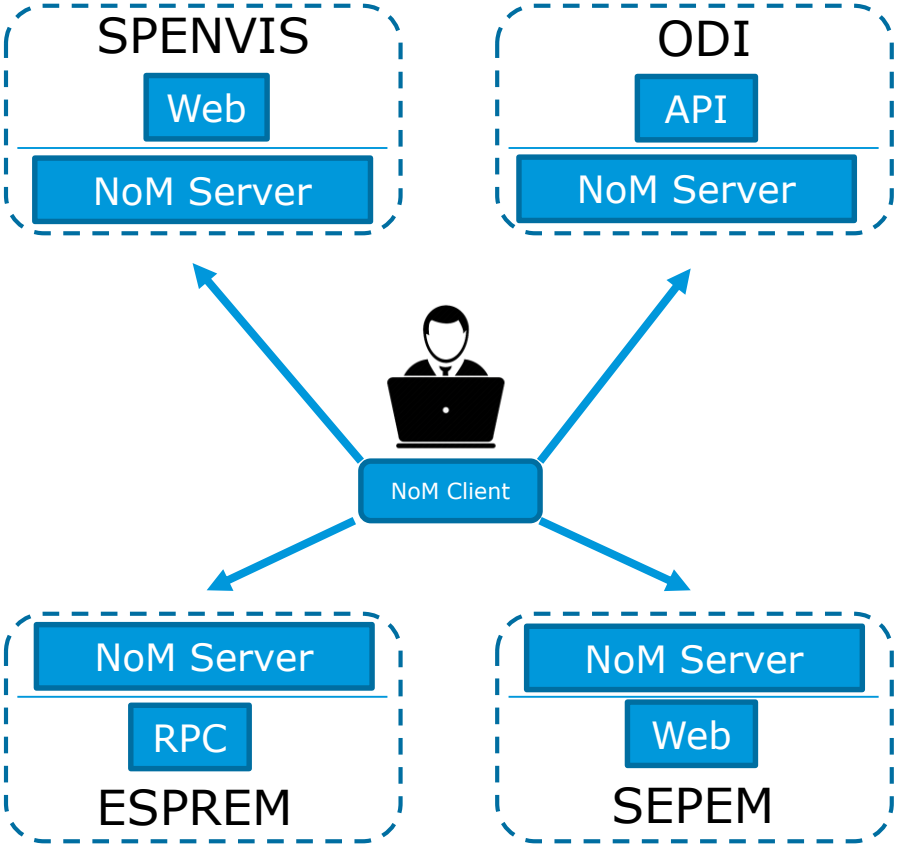
- Single-authoritative source of model information

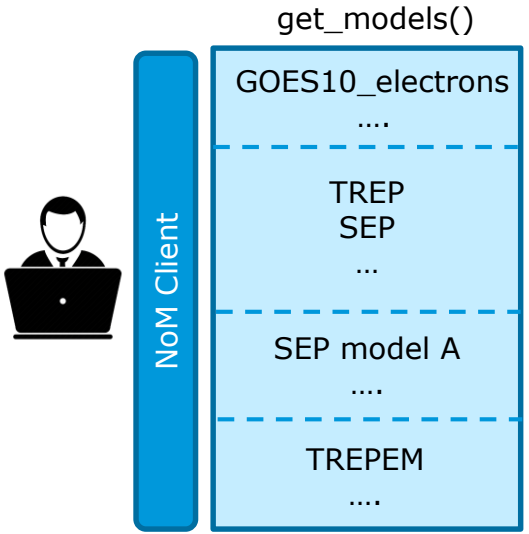
Standardise on data types:

- Spectra, time series, data maps, images …
- Particle fluxes, LET, dose-depth …

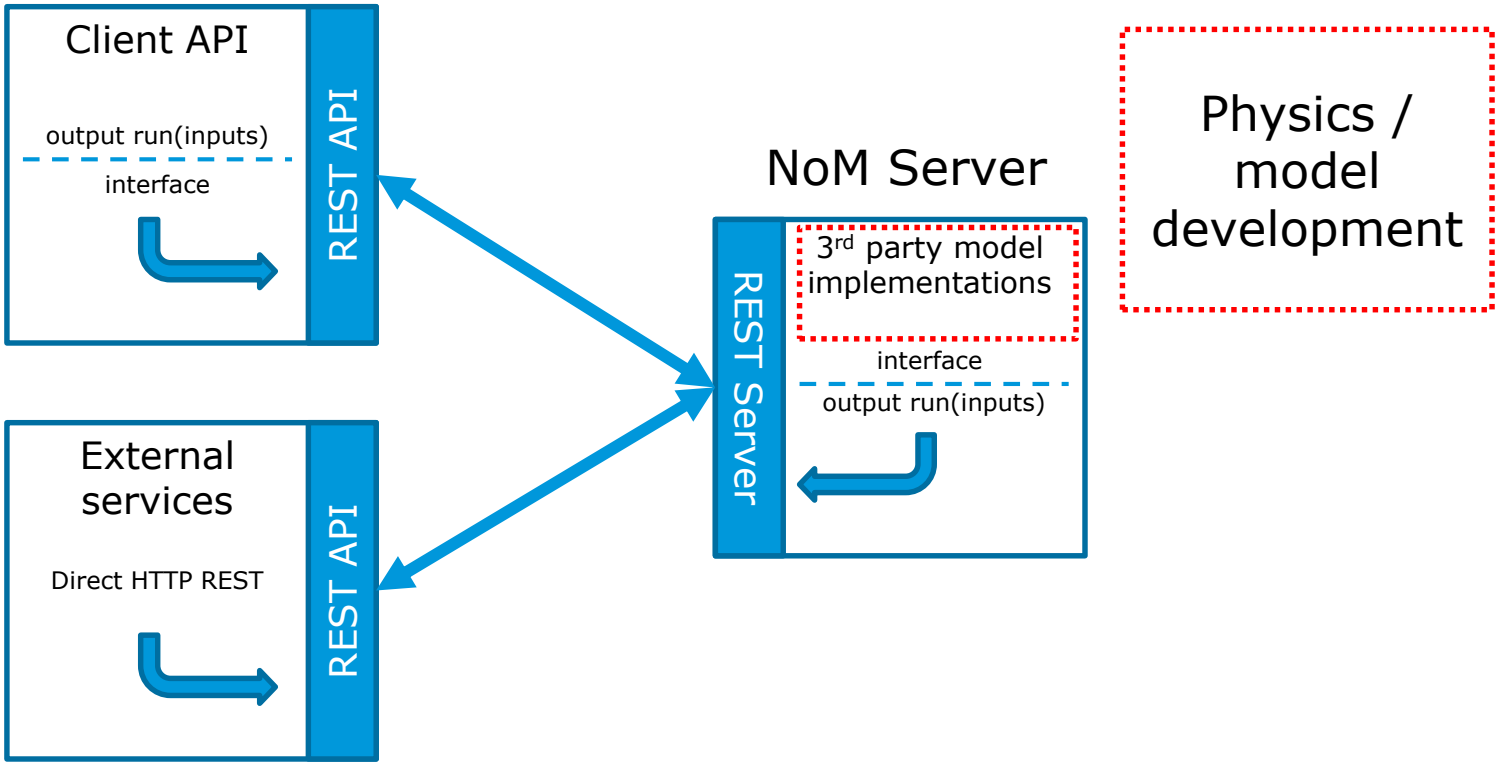
- Model interoperability and pipelining
- Simplifies data analysis
- Simplifies plotting
- …

Network of models concept: End user benefits





Network of Models Concept: Model developer benefits



Network of Models (NoM) Concept

Goals

Single client API to run all NoM models

- Easier discovery of models and data
- Less custom scripting and validating of data

Lightweight, general façade server to convert any model into a **NoM server**

- Can retro-fit existing projects
- Simple interface spec for new projects
- More time on model development
- Less requirement to create custom API

Standardise on model description:

- Model docs
- Model meta data (version, provider etc.)
- Input provision format
- Output formats
- Dynamic model GUI creation

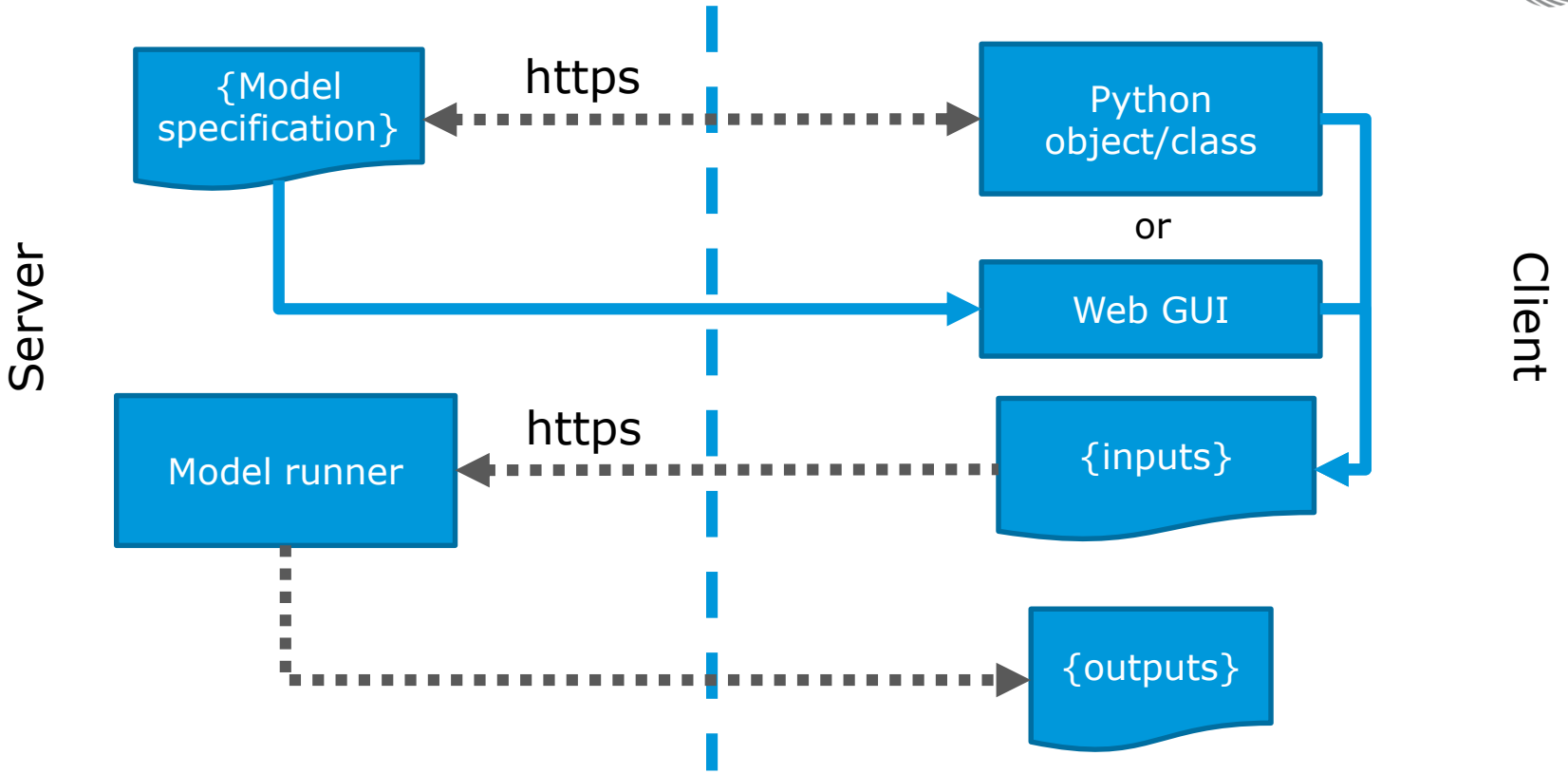
- Single-authoritative source of model information

Standardise on data types:

- Spectra, time series, data maps, images …
- Particle fluxes, LET, dose-depth …

- Model interoperability and pipelining
- Simplifies data analysis
- Simplifies plotting
- …

Network of Models concept: Model specifications



Network of Models concept: Model specification

```
{
  "meta": {
    "model_name": "trep",
    "category": "radiation/source/trapped"
  },
  "system": {
    "status": "include",
    "package": "models",
    "location": "/home/simon/spenvis/bin/",
    "binary": "trep",
    "modelProvider": "SpenvisImplementation",
    "outputReader": "SpenvisCSVReader",
    "provider": "server.thirdparty_impl.spenvis.spenvis_implementation",
    "output_reader": "server.thirdparty_impl.spenvis.spenvis_csv_reader"
  },
  "inputs": {
    "trep": [
      {"name": "trappedProtonModel", "type": "scalarInt", "required": "True", "default": 1, "valid_values": [1, 2, 3], "group": "AP-8", "model_mapping":
        "TRPMOD", "label": "Trapped proton model. [AP-8:1, SAMPEX/PET: 2, CRRESPRO: 3]"},
      ...
    ]
  },
  "requires": [
    {"data_type": "ORBIT"}
  ],
  "outputs": [
    {
      "type": "file",
      "file_ext": "tri",
      "outs": [
        [
          {
            "name": "orbit_average_proton_spectrum",
            "data_type": "SPECTRUM_ENERGY_FLUX_ISO",
            ...
          },
        ]
      ]
    }
  ]
}
```

Mapping, maps human friendly fields into model inputs

```
sapre_results = mrc.run_model(sapre_model, run_tag='orbit2')
print(sapre_results)

trep_model = mrc.get_model('trep')
trep_model.trappedProtonModel = 2
trep_result = mrc.run_model(trep_model, run_tag='orbit1')
print(trep_result)

sd2_model = mrc.get_model('sd2')
sd2_result = mrc.run_model(sd2_model, run_tag='orbit1')
print(sd2_result)
```

Allows client side, self-describing, self-documenting API for all models

Network of Models concept: Model specification / inputs

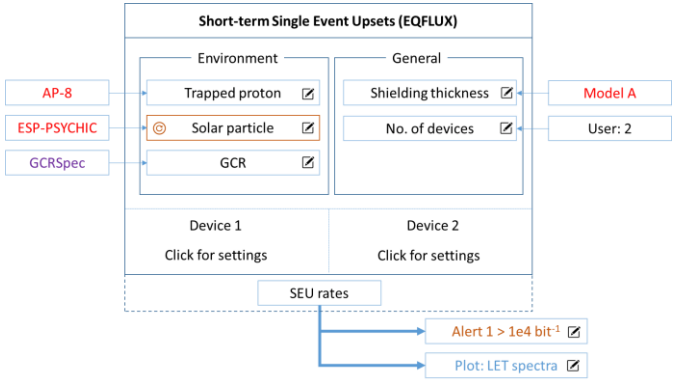
```
{
  "name": "trappedProtonModel",
  "model_mapping": "TRPMOD",
  "description": "Trapped proton model. [AP-8:1, SAMPEX/PET: 2, CRRESPRO: 3]",
  "type": "scalarInt",
  "required": "False",
  "default": 1,
  "valid_values": [
    1,
    2,
    3
  ],
  "depends_upon": {
    "input": "showProtonModels",
    "value": "True"
  },
  "view": {
    "page": 0,
    "grid": [1,2],
    "row_span": 1,
    "col_span": 1,
    "group": "AP-8",
    "label": "Trapped proton model"
  }
}
```

Mapping, maps human friendly fields into model inputs

Validation

Allows dynamic GUI

Basic GUI formatting



Network of Models concept: Model specification / outputs


```
"outputs": [
  {
    "type": "file",
    "file_ext": "tri.txt",
    "outs": [
      {
        "name": "orbit_average_proton_spectrum",
        "data_type": "FPDO",
        "columns":["ENERGY", "FPIO", "FPDO"],
        "split": "false",
        "species": "proton",
        "modifier": "orbit_average"
      },
      {
        "name": "orbit_average_electron_spectrum",
        "data_type": "FEDO",
        "columns":["ENERGY", "FEIO", "FEDO"],
        "split": "true",
        "species": "electron",
        "modifier": "orbit_average"
      }
    ]
  },
  {
    "type": "file",
    "file_ext": "spp.txt",
    "outs": [
      {
        "name": "peak_proton_flux_spectrum",
        "data_type": "FPDO",
        "columns":["ENERGY", "FPIO", "FPDO"],
        "species": "proton",
        "modifier": "peak"
      }
    ]
  },
```


How best to define model outputs.

Network of Models (NoM) Concept

Goals

Single client API to run all NoM models

- Easier discovery of models and data
- Less custom scripting and validating of data

Lightweight, general façade server to convert any model into a **NoM server**

- Can retro-fit existing projects
- Simple interface spec for new projects
- More time on model development
- Less requirement to create custom API

Standardise on model description:

- Model docs
- Model meta data (version, provider etc.)
- Input provision format
- Output formats
- Dynamic model GUI creation

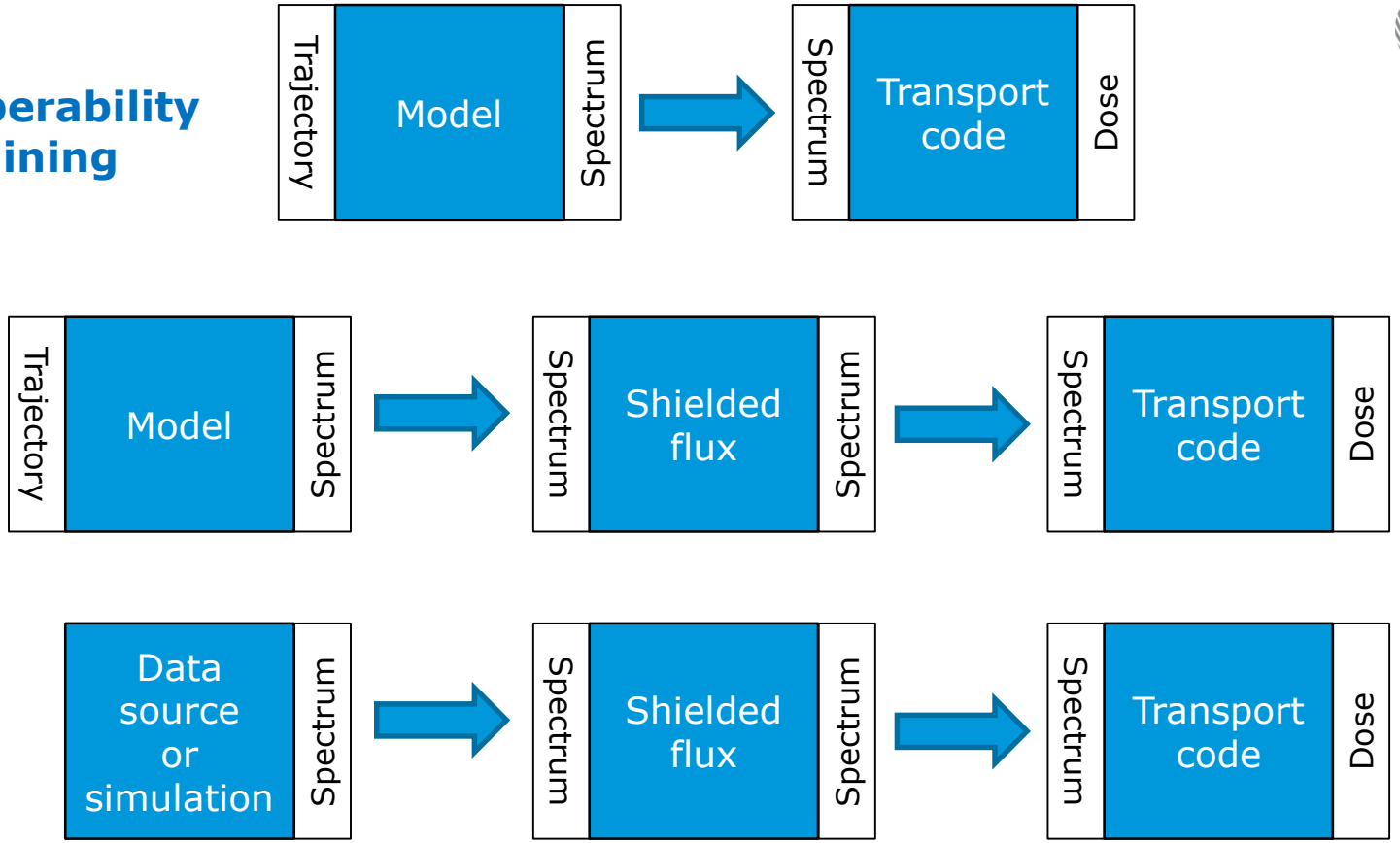
- Single-authoritative source of model information

Standardise on data types:

- Spectra, time series, data maps, images …
- Particle fluxes, LET, dose-depth …

- Model interoperability and pipelining
- Simplifies data analysis
- Simplifies plotting
- …

Model interoperability & pipelining



Network of Models: General requirements

- **Accessible**
  - Web, remote/RPC/API/RESTful access
- **Reliability**
  - High availability / discoverable availability / fallbacks / load-balancing
- **Version control**
  - The ability to specify the specific version of a model
- **Documented**
  - Online, easily accessible documentation for the model
  - Self-documenting models
- **Reproducibility**
  - Self-consistent outputs that include the inputs used to generate the outputs
- **User management / authorization**
  - Single-sign on
  - Token authentication and authorization / rate limiting
- **Secure**
  - Sanitisation

Any more?

Network of Models: Vocabulary

Model specification
Single authoritative description of a model

NoM Server
Python server to manage models and provide REST/RPC/etc interface

NoM Client
Python API for model discovery, running and results processing

Results Manager
Memory/file/SQL based results processor

Interfaces

Model provider
Interface to model specific code

Model reader
Model to model specific code to process model outputs

****These are the only files needed to be supplied by the model developer****

Example with SPENVIS, ESPREM & ODI

To enable SPENVIS as a **Model Provider**, only 2 files are required to be written:

spenvis_provider.py
- Handles the details of the namelist creation
- Runs the binaries

(~100 man-hours)

spenvis_csv_reader.py
- Reads the output files and provides them back to the framework

Exiting SPENVIS system does not need to be touched at all (**better if it is though**)

ESPREM and ODI were significantly easier as they have existing python APIs (~10 man-hours)

- Easy, robust interoperability of SPENVIS, ESPREM and ODI
- Iterative/parametric workflows

Thanks for listening

If you want to talk more about this please email me:

simon.clucas@esa.int