Extending SMRT

Towards a community model

Introduction

1 – Extending SMRT

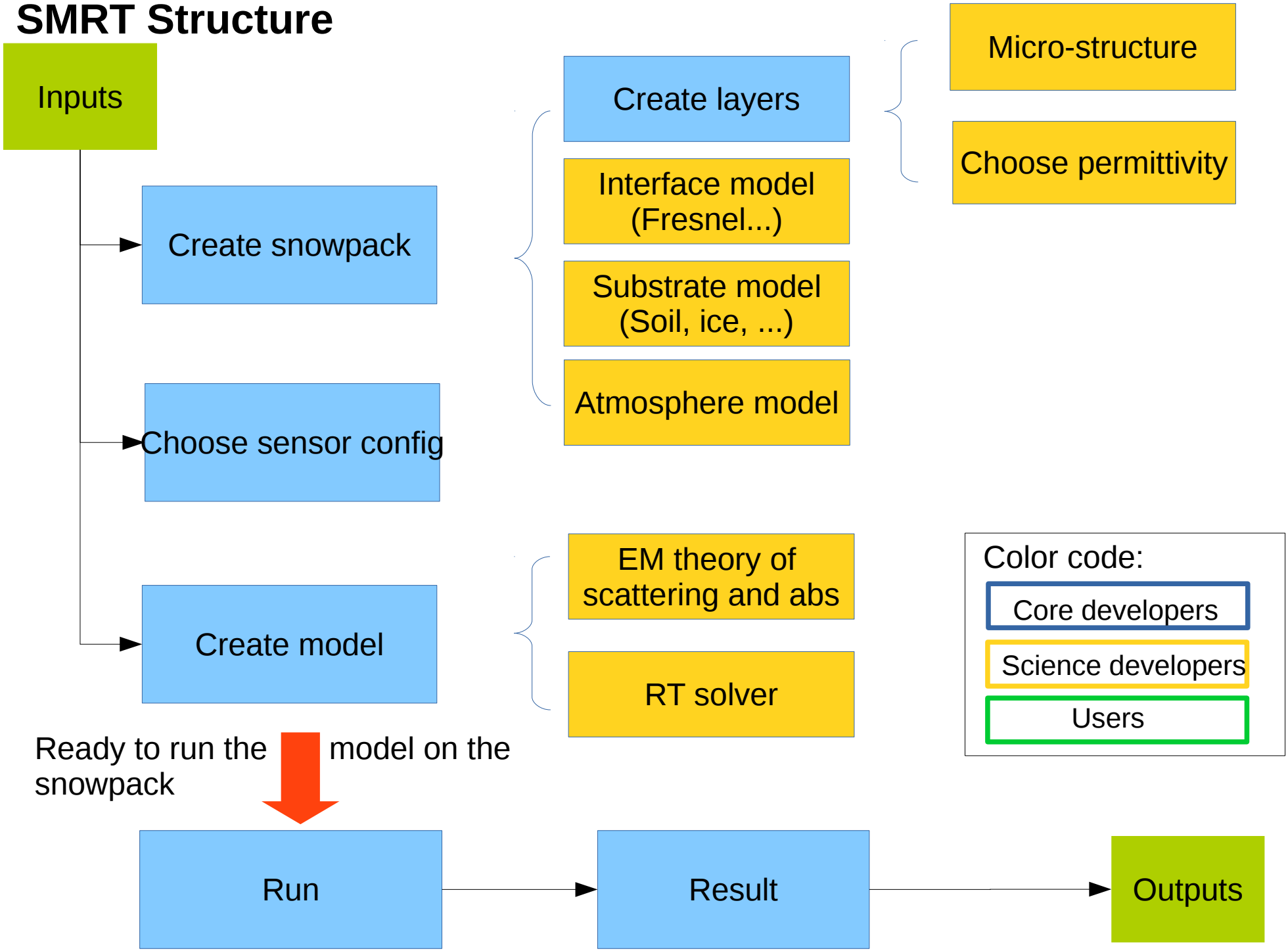
Point of view: I need a new permittivity formulation or EM theory or RT solver or microstructure, …

2 – Sharing developments

Point of view: I want to contribute to SMRT with my scientific devs

Science development

SMRT Structure



Science development

SMRT Structure ↔ directory structure

smrt/atmosphere	code to compute the Tbdown, trans et Tbup
smrt/core	don't change here! The main machinery but no science
smrt/emmodel	electromagnetic code IBA, DMRT, Rayleigh…
smrt/__init__.py	don't change here! were import start when « import smrt ».
smrt/inputs	user-oriented function to create snowpack, …
smrt/interface	code to compute R, T for inter-layer interfaces
smrt/microstructure_model	code with microstructure representation
smrt/permittivity	code with materials permittivity
smrt/rtsolver	code with RT Solvers
smrt/substrate	code to compute R, T for substrate
smrt/test	code to test smrt numerical results (using « nosetest »)
smrt/utils	various utilities related to smrt : wrappers to other Models, plotting functions, …

Science development

Recommended quick way to extend SMRT: Create a new file in the relevant directory, that's it !

E.g. to add a scattering theory:

1. Copy iba.py my_super_scatt_theory.py
2. Implement your change in my_super_scatt_theory.py
3. Ready to use and intercompare: m = make_model(« my_super_scatt_theory », « dort »)

No need to compile anything or create a configuration file. New files are automatically discovered.

Rmq :
Create new files, **do not modify existing files**.
-> Keep the compatibility : « git pull » works to get updates. Easy to transfer to someone, just email the new file and in which directory to put it. Your colleagues is ready to go !

Rmq :
To test variants : copy iba.py improved_iba.py, make the change, and
m1=make_model(« iba », « dort »)
m2=make_model(« improved_iba », « dort »)

I've optimized or developed most parts of SMRT like this, step by step keep a « reference » slow code and improve it in another file.

Science development

E.g.

Create a new file in the relevant directory, that's it !

E.g. to add a microstructure :

1. Copy exponential.py mysupermicrostructure.py
2. Edit mysupermicrostructure.py – **add your specific arguments**
3. Ready to use:

sp = make_snowpack(thickness, « mysupermicrostructure », ----------)

```
class Exponential(Autocorrelation):

    args = ["frac_volume", "corr_length"]
    optional_args = {}


class StickyHardSpheres(Autocorrelation):

    args = ["frac_volume", "radius"]
    optional_args = {"stickiness": np.inf}
```

Science development

Recommended GOOD way to extend SMRT:

Why/When is it better:

- You need versioning of your improvements (git, ...)
- You want to collaborate on improvements with others (likely through git)
- You have different improvements in //
- SMRT was installed as a package with pip install (docker, ...)

HowTo: Make a package independent from smrt that reproduces the directory structure of smrt

..../altim
..../altim/rtsolver/
..../altim/rtsolver/nadir_altimetry.py

And, just add this in your code:

register_package("altim")

(altim must be importable from python, that is the directory ../altim must in your PYTHONPATH)

Towards a community model

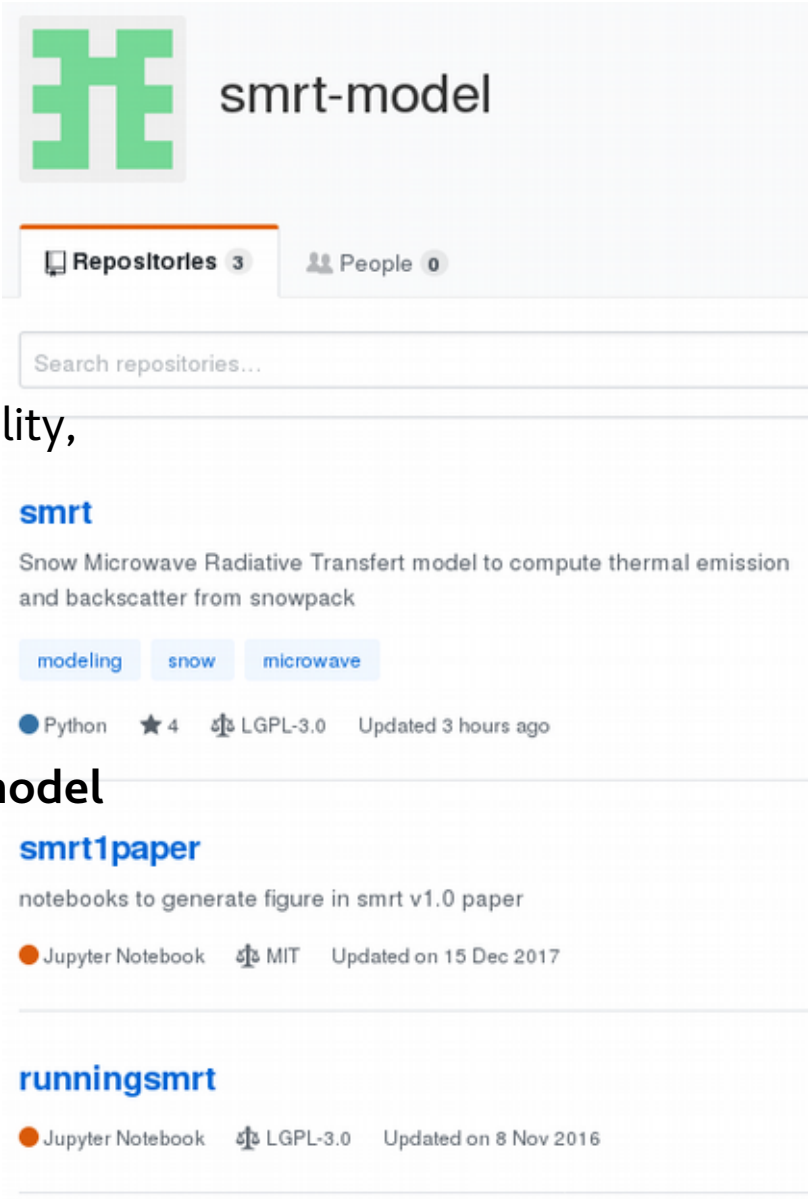
Sharing your scientific developments in SMRT is more than welcome, especially for published works.

Objective: Extend as much as possible
while maintaining quality

<u>Ideal requirements:</u>
- exactness and broad interest of the code
- clean code following guidelines and documentation
- sustainibility and a vision/roadmap (backward compatibility, no overlap, ...)

<u>Several levels of maturity:</u>
1- In a public repository on your own (github, ...)
2- In a “user-contrib” repository on **github smrt-model**
3- Integration in SMRT codebase itself on **github smrt-model**

1 and 2 work with « register_package »



Towards a community model

SMRT coding rules (for point 3):

Avoid scientific ambiguity, ensure future maintenance, make possible future development without breaking compatibility,

- explicit names for file, class and variable (lowercase word separated by _, except for classes, see PEP8). Names must be clear and non ambiguous. Almost no abbreviations are accepted. Short is better than long, but explicit is always better than implicit.

- make the functions and classes as general as possible + use option arguments with default values for the most widely “expected behavior”. « Beginner and advanced user friendly ».

- use S.I. unit without multiplictor or divisor: m, kg, s, Hz. No ambiguity.

- code formatted using PEP8 (with some rules relaxed).

- documentation directly in python code → autogenerated to readthedoc.io

- write unit test (files starting with test_) for every piece of code.

Towards a community model

Roadmap or how you can effectively help:

Sorted by increasing difficulty:
- read, comment and edit the online documentation. Adding refs, more explanations
- write tutorials or organize training
- add pre-defined sensors
- add permittivity formulations for ice and other materials (e.g. Turi's formulation)
- add soil models for passive (e.q. QNH model, see DMRT-ML)
- add HUT atmosphere or other simple model
- code review, writing unit test.

My personal roadmap:

- code optimization and // computing (in progress). See: Model.run_later and promise
- add RT solvers:
    1) Alitmetry (in progress).
    2) 6-flux (in progress).
    3) DORT with coherent layers (C. Matzler appraoch)
    4) solver for birefringent media
    5) recode how atmosphere is working

- improve IBA for 3-phase medium (of interest for sea-ice)

- AIEM for rough surface and rough layer interface (intiated).