Supplement of Earth Syst. Dynam., 11, 395–413, 2020
https://doi.org/10.5194/esd-11-395-2020-supplement


Earth System Dynamics
Open Access
EGU

*Supplement of*

# Earth system modeling with endogenous and dynamic human societies: the copan:CORE open World–Earth modeling framework

**Jonathan F. Donges et al.**

*Correspondence to:* Jonathan F. Donges (donges@pik-potsdam.de) and Jobst Heitzig (heitzig@pik-potsdam.de)

# Supplementary Information

## 1 The copan:CORE open World-Earth modeling framework – Details

In this section, we present the World-Earth modeling framework copan:CORE that was designed following the principles given in Sect. 1 of the main text in more detail than in Sect. 2 of the main text. Here we describe our framework on three levels, starting with the abstract level independent of any software (Sects. 1.2 and 1.3), then describing the software design independent of any programming language (Sect. 1.4), and finally presenting details of our reference implementation in the Python language (Sect. 1.5).

In summary, copan:CORE enables a flexible model design around standard components and model setups that allows investigation of a broad set of case studies and research questions (Fig. 1). Its flexibility and role-based modularization are realized within an object-oriented software design and support flexible scripting by end users and interoperability and dynamic coupling with existing models – e.g., the terrestrial vegetation model *LPJmL* working on the cell level (Bondeau et al., 2007) or other Earth system models or integrated assessment models based on time-forward integration (rather than intertemporal optimization) such as *IMAGE* (van Vuuren et al., 2015). On the level of model infrastructure, a careful documentation and software versioning via the 'git' versioning system aim to support collaborative and structured development in large teams using copan:CORE.

### 1.1 Features of the copan:CORE modeling framework

The *copan:CORE World-Earth modeling framework* presented in this paper is a code-based (rather than graphical) simulation modeling framework with a clear focus on Earth system models with complex human societies. It was developed within the flagship project 'copan – coevolutionary pathways' and will form the core of its further model development, which explains the naming. Similar to the common definition of 'software framework', we define a '(simulation) modeling framework' as a tool that provides a standard way to build and run simulation models.

We have designed copan:CORE to meet the special requirements for model development in the context of Earth system analysis: First, the framework's modular organization combines processes into model components. Different components can implement different, sometimes disputed, assumptions about human behavior and social dynamics from theories developed

within different fields or schools of thought. This allows for comparison studies in which one component is replaced by a different component modeling the same part of reality in a different way and exploring how the diverging assumptions influence the model outcomes. All components can be developed and maintained by different model developers and flexibly composed into tailor-made models used for particular studies by again different researchers. Second, our framework provides coupling capabilities to preexisting biophysical Earth system and economic integrated assessment models and thus helps to benefit from the knowledge of the detailed processes embedded in these models.

Finally, copan:CORE facilitates the integration of different types of modeling techniques. It permits for example to combine agent-based models (e.g., of a labor market at the micro-level of individuals) with systems of ordinary differential equations (modeling for example a carbon cycle). Similarly, systems of implicit and explicit equations (e.g., representing a multi-sector economy) can be combined with Markov jump processes (for example representing economic and environmental shocks).

These features distinguish the copan:CORE modeling framework from existing modeling frameworks and platforms. Before we continue with a more detailed description of the modeling framework, we go back to the underlying design principles of WEMs that guided the development of copan:CORE.

### 1.2 Abstract structure

This section describes the abstract structure of models that can be developed with copan:CORE and gives rationales for our design choices, many of which are based on experiences very similar to those reported in Robinson et al. (2018), in particular regarding the iterative process of scientific modeling and the need for open code, a common language, and a high level of consistency without losing flexibility.

#### 1.2.1 Entities, processes, attributes

A model composed with copan:CORE describes a certain part of the World-Earth system as consisting of a potentially large set (that may change over model time) of sufficiently well-distinguishable *entities* ("things that are", e.g., a spot on the Earth's surface, the European Union [EU], yourself). Entities are involved in a number of sufficiently well-distinguishable *processes* ("things that happen", e.g., vegetation growth, economic production, opinion formation). Processes in turn affect one or more *attributes* ("how things are", e.g., the spot's harvestable biomass, the EU's gross product, your opinion on fossil fuels, the atmosphere-ocean diffusion coefficient). During a model run, entities may come into existence (individuals may be born, social systems may merge into larger ones or fractionate), cease to exist (individuals may die, social systems may collapse), or may even be "reactivated" (e.g., an occupied country may regain independence).

**Rationale.** While for some aspects of reality an ontological distinction between entities, attributes of entities, and processes might be ambiguous, it corresponds very well to both the distinction of nouns, adjectives, and verbs in natural languages, and to the concepts of objects, object attributes, and methods in object-oriented programming.

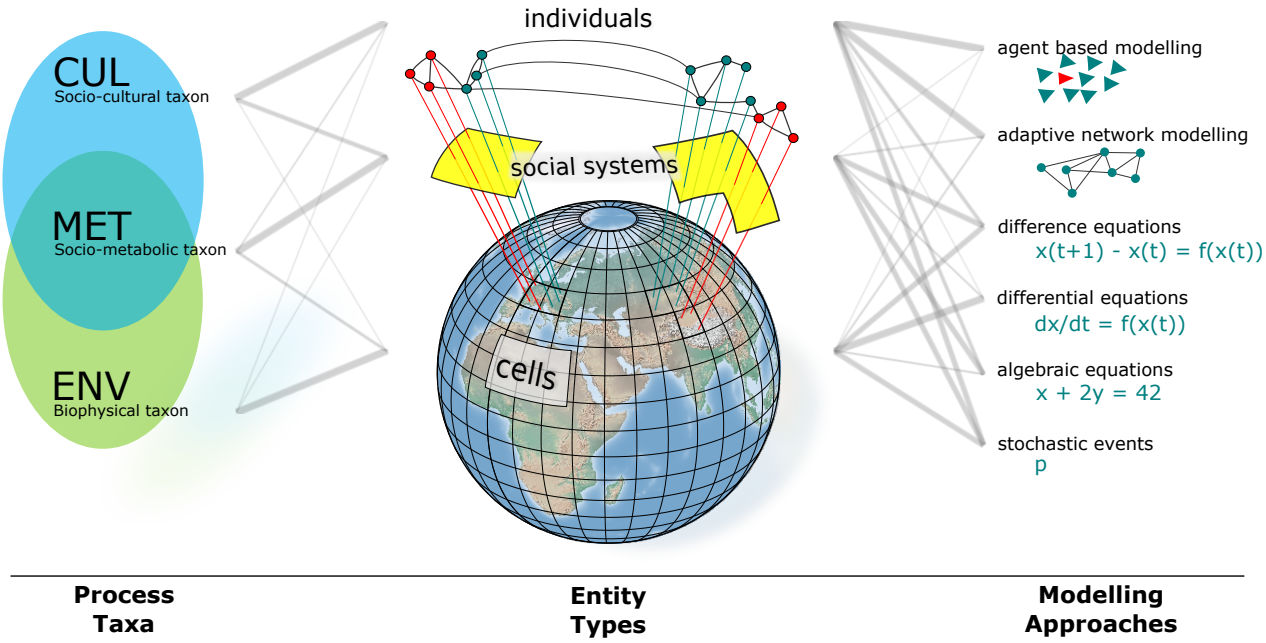


**Figure S1. Overview of copan:CORE open World-Earth modeling framework.** The entities in copan:CORE models are classified by *entity types* (e.g., grid cell, social system, individual, see middle column). Each process belongs to either a certain entity type or a certain *process taxon* (left column). Processes are further distinguished by formal process types (see text for a list) which allow for various different *modeling approaches* (right column). Entity types, process taxa and process types can be freely combined with each other (grey lines). Thick grey lines indicate which combinations are most common.

### 1.2.2 Entity types, process taxa, process types

copan:CORE classifies entities by *entity types* ("kinds of things that are", e.g., spatial grid cell, social system, individual), and allows to group (some or all) processes into *process taxa* (e.g., environmental, social-metabolic, socio-cultural). Each process and each attribute *belongs to* either a certain entity type or a certain process taxon. We deliberately do not specify criteria for deciding where processes belong since this is in part a question of style and academic discipline and there will inevitably be examples where this choice appears to be quite arbitrary and will affect only the model's description, implementation, and maybe its running time, but not its results.

Similarly, attributes may be modeled as belonging to some entity type (e.g., 'total population' might be modeled as an attribute of the 'social system' entity type) or to some process taxon (e.g., 'atmosphere-ocean diffusion coefficient' might be modeled as an attribute of the 'environment' process taxon). We suggest to model most quantities as entity type attributes and model only those quantities as process taxon attributes which represent global constants.

Independently of where processes belong to, they are also distinguished by their formal *process type,* corresponding to different mathematical modeling and simulation/solving techniques:

- continuous dynamics given by ordinary differential equations,
- (quasi-)instantaneous reactions given by algebraic equations (e.g., for describing economic equilibria),
- steps in discrete time (e.g., for processes aggregated at annual level or for coupling with external, time-step-based models or model components), or
- events happening at irregular or random time points (e.g., for agent-based and adaptive network components or externally generated extreme events).

the latter two potentially have probabilistic effects. Later versions will also include support for stochastic differential equations or other forms of time-continuous noise, currently noise can only be modeled via time-discretized steps. Similarly, attributes have *data types* (mostly physical or socio-economic simple quantities of various *dimensions* and *units,* but also more complex data types such as references or networks).

Fig. 1 summarizes our basic process taxa and entity types, their typical connections, and the corresponding typical modeling approaches (which in turn are related but not equal to certain formal process types, not shown in the figure). Sects. 1.3 describes them in detail.

**Rationale.** When talking about processes, people from very different backgrounds widely use a subject-verb-object sentence structure even when the subject is not a conscious being and the described action is not deliberate (e.g., "the oceans take up carbon from the atmosphere"). copan:CORE therefore allows modelers to treat some processes as if they were "done by" a certain entity (the "subject" of the process) "to" itself and/or certain other entities (the "objects" of the process). Other processes for which there appears to be no natural candidate entity to serve as the "subject" can be treated as if they are happening "inside" or "on" some larger entity that contains or otherwise supports all actually involved entities. In both cases, the process is treated as belonging to some entity type. Still other processes such as multilateral trade may best be treated as not belonging to a single entity and can thus be modeled as belonging to some process taxon.

A twofold classification of processes according to both ownership and formal process type is necessary since there is no one-to-one relationship between the two, as the grey lines in Fig. 1 indicate. E.g., processes from all three taxa may be represented by ODEs or via stochastic events, and all shown entity types can own regular time stepped processes.

### 1.2.3 Modularization, model components, user roles

copan:CORE aims at supporting a plug-and-play approach to modeling and a corresponding division of labour between several user groups (or *roles*) by dividing the overall model-based research workflow into several tasks. As a consequence, we formally distinguish between model components and (composed) models.

A *model component* specifies (i) a meaningful collection of processes that belong so closely together that it would not make much sense to include some of them without the others into a model (e.g., plants' photosynthesis and respiration), (ii) the entity attributes that those processes deal with, referring to attributes listed in the master data model whenever possible, (iii) which existing (or, if really necessary, additional) entity types and process taxa these processes and attributes belong to. A

*model* specifies (i) which model components to use, (ii) if necessary, which components are allowed to overrule parts of which other components (iii) if necessary, any *attribute identities*, i.e., whether some generally distinct attributes should be considered to be the same thing in this model (e.g., in a complex model, the attribute 'harvestable biomass' used by an 'energy sector' component as input may need to be distinguished from the attribute 'total vegetation' governed by a 'vegetation dynamics' component, but a simple model that has no 'land use' component that governs their relationship may want to identify the two).

The suggested workflow is then this:

- If there is already a model that fits your research question, use that one in your study (role: *model end user*).
- If not, decide what model components the question at hand needs.
  - If all components exist, compose a new model from them (role: *model composer*).
  - If not, design and implement missing model components (role: *model component developer*). If some required entity attributes are not yet in the master data model (Sect. 1.2.4), add them to your component. Suggest well-tested entity attributes, entity types, or model components to be included in the copan:CORE community's master data model or master component repository (*modeling board members* will then review them).

**Rationale.** Although in smaller teams, one and the same person may act in all of the above roles, the proposed role concept helps structuring the code occurring in a model-based analysis into parts needed and maintained by different roles, a prerequisite for collaborative modeling, especially across several teams.

The additional concept of model components (in addition to entity types and taxa) is necessary since processes which belong together from a logical point of view and are hence likely to be modeled by the same person or team may still most naturally be seen as being owned by different entity types, and at the same time developers from several teams may be needed to model all the processes of some entity type.

### 1.2.4 Master data model and master component repository

The *master data model* defines entity types, process taxa, attributes, and physical dimensions and units which the modeling board members deem (i) likely to occur in many different models or model components and (ii) sufficiently well-defined and well-named (in particular, specific enough to avoid most ambiguities but avoiding a too discipline-specific language). Users are free to define additional attributes in their components but are encouraged to use those from the master data model or suggest new attributes for it.

The *master component repository* contains model components which the modeling board members deem likely to be useful for many different models, sufficiently mature and well-tested, and indecomposable into more suitable smaller components. Users are free to distribute additional components not yet in the repository.

**Rationale.** Poorly harmonized data models are a major obstacle for comparing or coupling simulation models. Still, a perfectly strict harmonization policy that would require the prior approval of every new attribute or component would inhibit fast prototyping and agile development. This is why the above two catalogs and the corresponding role were introduced.

### 1.2.5 All attributes are treated as variables with metadata

Although many models make an explicit distinction between "endogenous" and "exogenous" variables and "parameters", our modular approach requires us to treat all relevant entity type or process taxon attributes a priori in the same way, calling them *variables* whether or not they turn out to be constant during a model run or are used for a bifurcation analysis in a study.

A variable's specification contains *metadata* such as a common language label and description, possibly including references to external metadata catalogs such as the Climate and Forecast Conventions' Standard Names (CF Standard Names, 2018) for climate-related quantities or the World Bank's CETS list of socio-economic indicators (World Bank CETS codes, 2017), a mathematical symbol, its level of measurement or scale of measure (ratio, interval, ordinal, or nominal), its physical or socio-economic dimension and default unit (if possible following some established standard), its default (constant or initial) value and range of possible values.

**Rationale.** The common treatment of variables and parameters is necessary because a quantity that one model component uses as an exogenous parameter that will not be changed by this component will often be an endogenous variable of another component, and it is not known to a model component developer which of the quantities she deals with will turn out to be endogenous variables or exogenous parameters of a model or study that uses this component. Well-specified metadata are essential for collaborative modeling to avoid hard-to-detect mistakes involving different units or deviating definitions.

## 1.3 Basic entity types

We try to keep the number of explicitly considered entity types manageably small and thus choose to model some relevant things that occur in the real world not as separate entities but rather as attributes of other entities. As a rule of thumb (with the exception of the entity type 'world'), only things that can occur in potentially large, a priori unknown, and maybe changing numbers and display a relevant degree of heterogeneity for which a purely statistical description seems inadequate will be modeled as entities. In contrast, things that typically occur only once for each entity of some type (e.g., an individual's bank account) or which are numerous but can sufficiently well described statistically are modeled as attributes of the latter entity type.

Although further entity types (e.g., 'household', 'firm', 'social group', 'policy', or 'river catchment') will eventually be included into the master data model, at this point the copan:CORE *'base' model component* only provides the entity types which all models must contain, described in this section, in addition to an overall entity type 'world' that may serve as an anchor point for relations between entities (see also Fig. 2).

### 1.3.1 Cells

An entity of type 'cell' represents a small spatial region used for discretising the spatial aspect of processes and attributes which are actually continuously distributed in space. They may be of a more or less regular shape and arrangement, e.g., represent a latitude-longitude-regular or an icosahedral grid or an irregular triangulation adapted to topography. Since they have no real-

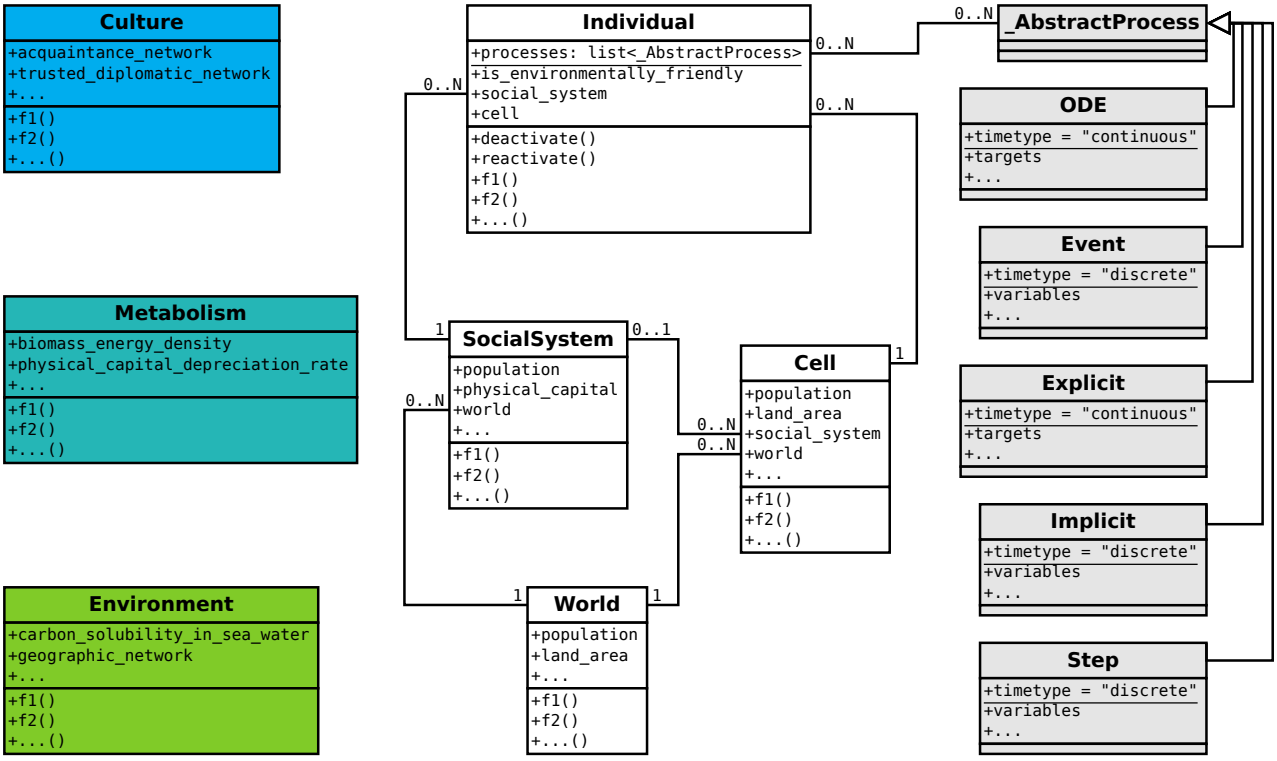


**Figure S2. Basic relationships between entities in the copan:CORE framework.** This UML class diagram shows the most important entity types and relationships, and a selection of entities' attributes, as implemented in the 'base' model component of the *pycopancore* reference implementation. 'f1()' and 'f2()' are placeholders for process implementation methods belonging to that taxon or entity type. The underlined attributes 'processes' (present in all taxa and entity types though shown only once here) and 'timetype' are class-level attributes.

world meaning beyond their use for discretization, cells are not meant to be used as agents in agent-based model components. Geographical regions with real-world meaning should instead be modeled via the type 'social system'.

### 1.3.2 Social systems

An entity of type 'social system' is meant to represent what is sometimes simply called a 'society', i.e. "an economic, social, industrial or cultural infrastructure" (Wikipedia, 2017) such as a megacity, country, or the EU. We understand a social system as a human-designed and human-reproduced structure including the flows of energy, material, financial and other resources that are used to satisfy human needs and desires, influenced by the accessibility and usage of technology and infrastructure (Fischer-Kowalski, 1997; Otto et al., 2020). Equally importantly, social systems include social institutions such as informal systems of norms, values and beliefs, and formally codified written laws and regulations, governance and organizational structures (Williamson, 1998). In our framework, norms, values and beliefs may be described in macroscopic terms on the social system level but may also be described microscopically on the level of individuals (Sect. 1.3.3).

Social systems in this sense typically have a considerable size (e.g., a sovereign nation state such as the United States of America, a federal state or country such as Scotland, an urban area such as the Greater Tokyo Area, or an economically very closely integrated world region such as the EU), controlling a well-defined territory (represented by a set of cells) and encompassing all the socio-metabolic and cultural processes occurring within that territory. Social systems are not meant to represent a single social group, class, or stratum, for which different entity types should be used (e.g., a generic entity type 'social group'). To allow for a consistent aggregation of socio-metabolic quantities and modeling of hierarchical political decision-making, the social systems in a model are either all disjoint (e.g., representing twelve world regions as in some integrated assessment models, or all sovereign countries), or form a nested hierarchy with no nontrivial overlaps (e.g., representing a three-level hierarchy of world regions, countries, and urban areas). As the attributes of social systems will often correspond to data assembled by official statistics, we encourage to use a set of social systems that is compatible to the standard classification ISO 3166-1/2 when representing real-world social systems.

Social systems may act as agents in agent-based model components but an alternative choice would be to use 'individuals' like their 'head of government' or 'social groups' like a 'ruling elite' as agents.

### 1.3.3 Individuals

Entities of type 'individual' represent individual human beings. These entities will typically act as agents in agent-based model components, although also entities of other types (e.g., the potential types 'household', 'firm', or 'social group') may do so. In contrast to certain economic modeling approaches that use "representative" consumers, an entity of type 'individual' in copan:CORE is not usually meant to represent a whole class of similar individuals (e.g., all the actual individuals of a certain profession) but just one specific individual. Still, the set of all 'individuals' contained in a model will typically be interpreted as being a representative *sample* of all real-world people, and consequently each individual carries a quantity 'represented population' as an attribute to be used in statistical aggregations, e.g., within a social system.

### 1.3.4 Relationships between entity types and process taxa

Although there is no one-to-one correspondence between process taxa and entity types, some combinations are expected to occur more often than others, as indicated by the thicker gray connections in Fig. 1.

We expect processes from the *environmental (ENV)* process taxon to deal primarily with the entity types 'cell' (for local processes such as terrestrial vegetation dynamics described with spatial resolution) and 'world' (for global processes described without spatial resolution, e.g., the greenhouse effect) and sometimes 'social system' (for mesoscopic processes described at the level of a social system's territory, e.g., the environment diffusion and decomposition of industrial wastes).

*Socio-metabolic (MET)* processes are expected to deal primarily with the entity types 'social system' (e.g., for processes described at national or urban level), 'cell' (for local socio-metabolic processes described with additional spatial resolution for easier coupling to natural processes) and 'world' (for global socio-metabolic processes such as international trade), and only rarely with the entity type 'individual' (e.g., for micro-economic model components such as consumption, investment or the job market).

Finally, processes from the *socio-cultural (CUL)* taxon are expected to deal primarily with the entity types 'individual' (for "micro"-level descriptions) and 'social system' (for "macro"-level descriptions), and rarely 'world' (for international processes such as diplomacy or treaties).

### 1.4 Software design

This section describes the programming language-independent parts of how the above abstract structure is realized as computer software. As they correspond closely with the role-based and entity-centric view of the abstract framework, *modularization* and *object-orientation* are our main design principles. All parts of the software are organized in packages, subpackages, modules, and classes. The only exception are those parts of the software that are written by model end-users to perform actual studies, which will typically be in the form of *scripts* following a mainly imperative programming style that uses the classes provided by the framework. Fig. 3 summarizes the main aspects of this design which are described in detail in the following.

#### 1.4.1 Object-oriented representation

Entity types and process taxa are represented by *classes* ('Cell', 'SocialSystem', 'Culture', ...), individual entities by *instances* (objects) of the respective entity type class, and process taxon classes have exactly one instance. While entity type and process taxon classes hold processes' and variables' metadata as *class attributes,* entity instances hold variable values and, where needed, their time derivatives as *instance attributes.* Processes' logics can be specified via *symbolic expressions* in the process metadata (e.g., for simple algebraic or differential equations) or as imperative code in *instance methods* (e.g., for regular 'steps' and random 'events' in an agent-based modeling style), thereby providing a large flexibility in how the equations and rules of the model are actually represented in the code, without compromising the interoperability of model components.

#### 1.4.2 Interface and implementation classes

All of this is true not only on the level of (composed) models but already on the level of model components, though restricted to the entity types, processes and variables used in the respective component. To avoid name clashes but still be able to use the same simple naming convention throughout in all model components, each model component is represented by a *subpackage* of the main copan:CORE software package, containing class definitions for all used entity types and process taxa as follows. Each entity type and process taxon used in the model component is represented by two classes, (i) an *interface class* that has a class attribute of type 'Variable' (often imported from the master data model subpackage or another model component's interface classes) for each variable of this entity type or process taxon this model component uses as input or output, containing that variable's metadata (see Fig. 1 in the Supplementary Information for an example), and (ii) an *implementation class* inherited from the interface class, containing a class attribute 'processes' and potentially some instance methods with process logics.

The attribute 'processes' is a list of objects of type 'Process', each of which specifies the metadata of one process that this model component contributes to this entity type or process taxon (see Figs. 2 and 3 of the Supplementary Information for examples). These metadata either contain the process logics as a symbolic expression or as a reference to some instance

method(s). Instance methods do not return variable values but manipulate variable values or time derivatives directly via the respective instance attributes. As many variables are influenced by more than one process, some process implementation methods (e.g., those for differential equations or noise) only add some amount to an attribute value, while others (e.g., those for major events) may also overwrite an attribute value completely.

#### 1.4.3 Model composition via multiple inheritance

Finally, a model's composition from model components is represented via multiple *inheritance* from the model component's implementation classes (which are thus also called 'mixin' classes) as follows. Each model is defined in a separate *module* (typically a single code file). For each entity type and process taxon that is defined in at least one of the used model component packages, the model module defines a composite class that inherits from all the mixin classes of that entity type contained in the used model component packages. Fig. 3 shows an example of this with just two components and two entity types.

#### 1.4.4 Dimensional quantities, symbolic expressions, networks

To be able to specify values of dimensional quantities, mathematical equations, and networks of relationships between entities in a convenient and transparent way, we provide classes representing these types of objects, e.g., 'Dimension', 'Unit', 'DimensionalQuantity', 'Expr' (for symbolic expressions), 'Graph' (for networks), 'ReferenceVariable'/'SetVariable' (for references to single/sets of other entities).

#### 1.4.5 Interoperability with other model software

copan:CORE can be used together with other simulation software to simulate coupled models consisting of "internal" components implemented in copan:CORE interacting in both directions with an "external" component provided by the other software. Currently, copan:CORE must act as the coupler to achieve this, which requires that the other software provides at least a minimal interface (e.g., conforming to the basic modeling interface (BMI), Syvitski et al. (2014)) that allows to read, set and change its state variables and to advance its model simulation by one time step.

To couple an external model component into a copan:CORE model, one must write a "wrapper" model component in the copan:CORE framework. For each relevant 'external' variable of the external model, the wrapper specifies a corresponding 'internal' copan:CORE variable in a suitable entity type or process taxon. In addition, the wrapper contributes a process implementation method of type 'Step' to a suitable process taxon, which uses the external software's interface to sync the external variables with their internal versions, using a suitable regridding strategy if necessary, and lets the external model perform a time step.

In later versions, copan:CORE will include a standard wrapper template for models providing a BMI, and might also itself provide such an interface to external couplers.

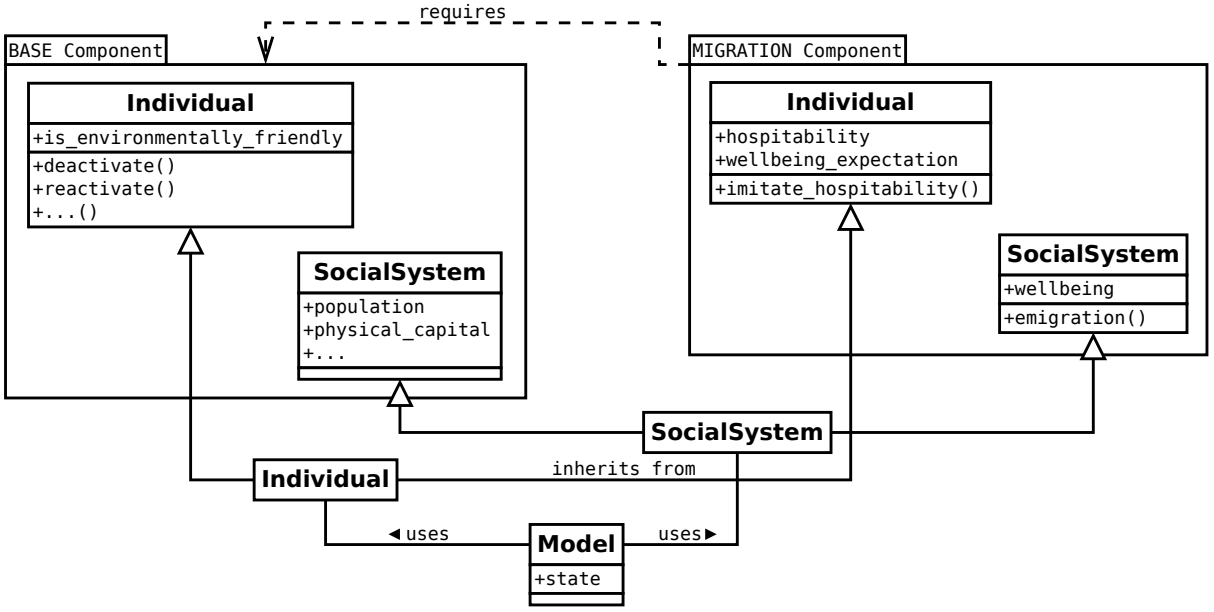


**Figure S3. Model composition through multiple inheritance of attributes and processes by process taxa and entity types.** This stylized class diagram shows how a model in copan:CORE can be composed from several model components (only two shown here, the mandatory component 'base' and the fictitious component 'migration') that contribute component-specific processes and attributes to the model's process taxa and entity types (only two shown here, 'Individual' and 'SocialSystem'). To achieve this, the classes implementing these entity types on the model level are composed via multiple inheritance (solid arrows) from their component-level counterparts (so-called 'mixin' classes).

### 1.5 Reference implementation in Python

For the reference implementation of copan:CORE we chose the Python programming language to enable a fast development cycle and provide a low threshold for end users. It is available as the open-source Python package *pycopancore* (`https://github.com/pik-copan/pycopancore`) including the master data model and a small number of pre-defined model components and models as subpackages and modules. Symbolic expressions are implemented via the *sympy* package (Meurer et al., 2017) which was extended to support aggregation (as in Fig. 3 of the Supplementary Information, top, line 5) and cross-referencing between entities (same Fig., bottom, line 14). ODE integration is currently implemented via the *scipy* package (Jones et al., 2001). While the reference implementation is suitable for moderately sized projects, very detailed models or large-scale Monte-Carlo simulations may require an implementation in a faster language such as C++, which we aim at realizing via a community-driven open-source software development project. Fig. 4 gives an impression of how user code in *pycopancore* looks like. See the *Supplementary Information* for further details.

**Model component developers** add their model component as a package to their local workspace folder, including the interface module and one module for each provided implementation class as in Figs. 5–7. In order to paralellize complex computations or couple to other model software, an implementation class may implement certain processes using an instance method

```
1  import pycopancore.models.my_model as M  # the model to be used
2  from pycopancore import master_data_model as D  # needed for dimensional quantities
3  from pycopancore.runners import DefaultRunner
4  # other imports
5
6  # instantiate the model, its process taxa, and some entities:
7  mod = M.Model()
8  world = M.World(environment=M.Environment(), metabolism=M.Metabolism(), culture=M.Culture(),
9                  atmospheric_carbon = 830 * D.gigatonnes_carbon)  # non-default initial value
10 socs = [M.SocialSystem(world=world) for s in range(10)]
11 cells = [M.Cell(socialsystem=random.choice(socs)) for c in range(100)]
12 inds = [M.Individual(cell=random.choice(cells),  # place individuals randomly on cells
13                      supports_emissions_tax = random.choice([False, True], p=[.7, .3]),
14                      imitation_rate = 1 / D.weeks)
15         for i in range(1000)]
16
17 # form an Erdos-Renyi random acquaintance network:
18 for index, i in enumerate(inds):
19     for j in inds[:index]:
20         if random.uniform() < 0.1: world.culture.acquaintance_network.add_edge(i, j)
21
22 # distribute initial global vegetation randomly among cells:
23 r = random.uniform(size=100)
24 M.Cell.terrestrial_carbon.set_values(cells, 2480 * D.gigatonnes_carbon * r / sum(r))
25
26 # run model and plot some results:
27 runner = DefaultRunner(model=mod)
28 traj = runner.run(t_0=2000, t_1=2200, dt=1)  # returns a dict of dicts of time-series
29 pylab.plot(traj[D.time], traj[M.World.surface_air_temperature][world], "r")
30 for s in socs: pylab.plot(traj[D.time], traj[M.SocialSystem.population][s], "y")
```

**Figure S4. Sketch of a model end user's Python script running a model and plotting some results**, featuring dimensional quantities and a network. Variable values can be set either at instantiation (line 9), via the entity object attribute (line 20) or the Variable object (line 24).

that calls or communicates with other processes or external programs which provide a communication method supported by Python, e.g., MPI or JSON.

**Model composers** provide a module that mainly composes the final entity type and process taxon classes via multiple inheritance from model components' implementation classes, e.g., specifying code like

```
import .climate_policy as pol
import pycopancore.model_components.simple_economy as econ
import pycopancore.base
class SocialSystem (pol.SocialSystem, econ.SocialSystem, base.SocialSystem):
    pass
```

**Model end users** use a Python script that imports these model modules, instantiates a 'model' object, all needed process taxon objects and an initial set of entities, then initializes those variables that shall start with non-default values, uses a 'runner' object to run the model for a specified time and finally analyses the resulting trajectory. Fig. 5 of the main text gives a sketch of such a script (see the online tutorial for more detailed examples).

Upon instantiation, the 'model' object uses Python's *introspection* capabilities to analyse its own model structure including which variables depend on which others in which way, and this information is then used by the runner to simulate the model. Future versions will use this information further for improving performance and producing reports on model structure. The runner returns the time evolution of requested variables as a nested Python dictionary the first- and second-level keys of which

Module pycopancore.model_components.**simple_policy_process**.interface:

```
1  from pycopancore import master_data_model as D
2  from pycopancore import Variable, BooleanVariable
3
4  class Model:  # mixin class holding model component's metadata
5      name = "simple policy process"
6      description = "At regular intervals, social systems set an emissions tax " \
7                    "if a majority of their population supports it. Individuals " \
8                    "imitate the corresponding attitude from their acquaitances."
9      requires = []
10
11 class SocialSystem:
12     # endogenous (output) variables:
13     emissions_tax = D.S.emissions_tax  # use a variable from the master data model
14     # exogenous (input) variables / parameters:
15     voting_period = Variable("voting period",  # defines a new variable
16                              unit=D.years, strict_lower_bound=0, default=4)
17     # (other)
18
19 class Individual:
20     # endogenous (output) variables:
21     supports_emissions_tax = BooleanVariable(  # subclass of Variable
22         "supports emissions tax",
23         desc="whether individual supports an emissions tax", default=False)
24     # exogenous (input) variables / parameters:
25     imitation_rate = Variable("imitation rate", unit = 1 / D.years, lower_bound=0)
26     # (other)
```

**Figure S5.** Sketch of a model component's interface, implemented as a Python module that lists the variables. The component contributes to the various entity types and process taxa, either referenced from the master data model (line 13) or defined newly (line 15).

are a 'variable' object and an entity or process taxon and whose values are lists of values ordered by time, which can then conveniently be analysed or plotted (e.g., Fig. 5 of the main text, line 30).

## 2 Details and potential extensions of the example model

For variables and parameters taken from Nitzbon et al. (2017) we chose values equivalent to those given in Table 1 of that paper, mainly except for $l_0$, which was chosen higher to accommodate for our additional space competition factor and make the initial photosynthesis flow fit current amounts. As initial conditions, we used rough global aggregates where data was available.

The model presented in the main text can easily be extended to include well-being-driven population growth and migration, renewable technology knowledge spillovers, and carbon taxation, for all of which the reference implementation shippes with corresponding model components.

### 2.1 Population growth

Like in Nitzbon et al. (2017), but here again on the `SocialSystem` level, population has a wellbeing-dependent fertility rate that was roughly fitted against country-level data of fertility vs GDP per capita. We chose the functional form $\mathrm{fert}_s = p_0 + 2(p-p_0)W_s W_P^{\omega_p} / (W_s^{1+\omega_p} + W_P^{1+\omega_p})$, where wellbeing $W_s = w_Y(1-i)Y_s/P_s + w_L L_s/\Sigma_s$ depends on per-capita consumption

Module pycopancore.model_components.**simple_network_adaptation**.implementation.culture:

```
# (imports)
class Culture (I.Culture):  # process taxon impl. mixin inheriting from interface

    def close_triads(self, unused_t):  # method performing the "triadic closure" step
        for i in self.individuals:  # i introduces two of her friends
            nbs = self.culture.acquaintance_network.neighbors(i)
            if len(nbs) > 1:
                j1 = random.choice(nbs); nbs.remove(j1); j2 = random.choice(nbs)
                if not G.has_edge(j1, j2): G.add_edge(j1, j2)

    processes = [  # list of processes contributed to Culture by this component
        Step("triadic closure",  # type and name of process
             [I.Culture.acquaintance_network],  # affected variable(s)
             1 * D.week, close_triads), # time step, method performing the step
        # (other processes)
    ]
```

Module pycopancore.model_components.**simple_policy_process**.implementation.individual:

```
# (imports)
class Individual (I.Individual):  # entity type implementation mixin class

    def imitate(self, unused_t):  # method performing the "imitation" event
        other = random.choice(self.culture.acquaintance_network.neighbors(self))
        self.support_emissions_tax = other.support_emissions_tax

    processes = [
        Event("imitation of attitudes",  # an irregularly timed discrete-time process
              [I.Individual.support_emissions_tax],  # affected variable(s)
              "rate", I.Individual.imitation_rate, imitate),  # subtype, rate, method
        # (other processes)
    ]
```

Module pycopancore.model_components.**simple_policy_process**.implementation.social_system:

```
# (imports)
class SocialSystem (I.SocialSystem):

    def take_a_vote(self, unused_t):
        vote_share = average([i.supports_emissions_tax for i in self.individuals],
                             weights=[i.represented_population for i in self.individual
        self.emissions_tax = self.culture.emissions_tax_level if vote_share > 0.5 else

    processes = [
        Step("voting on emissions tax",
             [I.SocialSystem.emissions_tax], I.SocialSystem.voting_period, take_a_vote)
        # (other explicit and implict equations and ODEs)
    ]
```

**Figure S6.** Sketches of implementation classes for three entity-types in two model components, to be used as mixin classes in model compo- sition. Each class defines processes (here steps and events) that the owning model component contributes to a certain process taxon or entity type. Note how the examples feature process implementation via instance methods (l.4 of each example) networks (top, l.6–9), dimensional quantities (top, l.14), stochasticity (middle, l.5), and the use of a social system's individuals as a representative sample of its population (bottom, l.5+6). See inline comments in magenta for detailed explanations.

Module pycopancore.model_components.**simple_economy**.implementation.social_system:

```
# (imports)
class SocialSystem (I.SocialSystem):
  processes = [
    Explicit("fossil reserves", [I.SocialSystem.fossil_reserves],
      # use aggregation keyword "sum" on set-valued reference SocialSystem.cells:
      [B.SocialSystem.sum.cells.fossil_reserves]),
    Implicit("energy market price equilibrium",  # an implicit equation
      [I.SocialSystem.fossil_price, I.SocialSystem.renewables_price],
# used variables
      [sympy.Eq(I.SocialSystem.fossils_price + I.SocialSystem.emissions_tax,
                I.SocialSystem.renewables_price)]),  # equation given as a sympy equati
    # (other explicit and implict equations and ODEs)
  ]
```

Module pycopancore.model_components.**simple_carbon_cycle**.implementation.cell:

```
from pycopancore import Explicit, ODE  # needed process types
import pycopancore.base.interface as B  # base component defines refs. betw. entities &
from .. import interface as I  # this component's interface

class Cell (I.Cell):

  balance = (I.Cell.photosynthesis_carbon_flow              # a utility
             - I.Cell.terrestrial_respiration_carbon_flow)  # symbolic expression

  processes = [

    Explicit("respiration flow",  # an explicit equation calculating one variable
             [I.Cell.terrestrial_respiration_carbon_flow],  # target variable
             # get correct parameter value by following refs. between entities and
             # process taxa, here: from Cell to World to Environment:
             [(B.Cell.world.environment.basic_respiration_rate
               + B.Cell.world.environment.respiration_sensitivity_on_temperature
                 * B.Cell.world.surface_air_temperature)
              * I.Cell.terrestrial_carbon]),  # symbolic expr. for calc. target variabl

```

**Figure S7.** Sketches of implementation classes (continued), featuring explicit and implicit equations (top, l.4–10) and ODEs (bottom, l.23–26), symbolic expressions (bottom, l.7–8) and equations (top, l.9–10), aggregation (top, l.6), and cross-referencing between entities (bottom,l.16–18).

$(1-i)Y_s/P_s$ and the mean terrestrial carbon density in that social system, $L_s/\Sigma_s$. For small $W_s$, $f$ grows linearly, reaching its maximum at $W_s = W_P$, then decaying towards $p_0$ with an asymptotically power-law shape with exponent $\omega_p$ for large $W_s$. Similarly, for mortality we roughly fitted the function $q/(W_s/W_P)^{\omega_q}$ against data and added a term representing increased mortality from climate change impacts, $q_T(T - T_q)$, and one representing competition for space, $q_C P_s/\Sigma_s\sqrt{K_s}$, where the factor $1/\sqrt{K_s}$ represents the assumption that housing is a form of physical capital with decreasing marginal value.

Note that while population $P_s$ changes over time, the number of representative individuals in $s$ remains constant in our example model, implying that the share of the population in $s$ that a certain `Individual` $i$ represents will change over time. A more elaborate model could try to keep the ratio of population and number of representative individuals roughly constant by generating or deactivating instances of `Individual` in $s$ at the current birth and death rate of $s$.

### 2.2 Wellbeing-driven migration

In addition to births and deaths, `SocialSystems`' populations change due to migration depending on differences in wellbeing. We assume each person in $s$ has a probability of emigrating to $s'$ that is proportional to the available information about differences in wellbeing for which we use the population in $s''$ as a proxy. We also assume that the probability of migration depends on a sigmoidal function of the wellbeing ratio, $f(\log W_{s'} - \log W_s)$ with $f(-\infty) = 0$ and $f(\infty) = 1$. More specifically, the absolute emigration flow from $s$ to $s'$ is $\mu P_s P_{s'}(1/2 + \arctan(\pi\phi(\log W_{s'} - \log W_s - \log\rho))/\pi)$, where $\mu$ is a basic rate and $\phi$ and $\rho$ are slope and offset parameters.

### 2.3 Carbon taxation

In the general elections of a `SocialSystem`, also a GHG emissions tax may be introduced, leading to a shift in the energy price equilibrium of

marginal production cost of fossil energy + emissions tax

= marginal production cost of biomass energy + emissions tax

= marginal production cost of renewable energy − renewable subsidy.

### 2.4 Necessary improvements

We'd like to repeat that the example model was designed to showcase the concepts and capabilities of copan:CORE in a rather simple WEM, and its components were chosen so that all entity types and process taxa and most features of copan:CORE are covered. The example model is not intended to be a serious representation of the real world that could be used directly for studying research questions, and the shown time evolutions may not be interpreted as any kind of meaningful quantitative prediction or projection.

To develop the example model into a serious World-Earth model, very many things remain to be done, including a careful selection of processes to include or exclude, improvements in model equations and agent's behavioural rules, both by fitting data where available and adopting model components from the literature, suitable choice of real-world social systems to include as entities, appropriate gridding of the surface into cells, and a solid estimation of parameters and initial conditions and their local and societal differences.

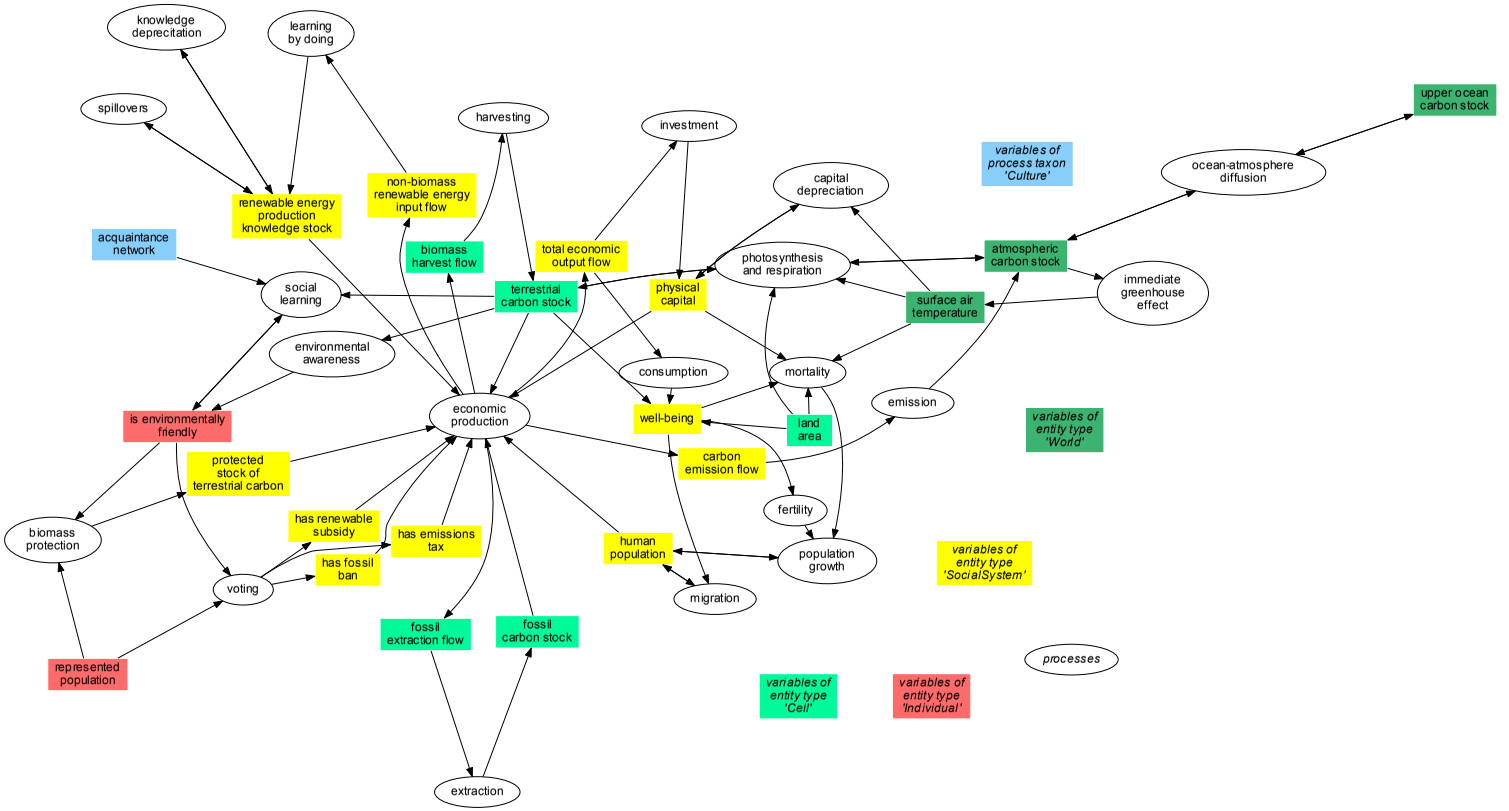


**Figure S8.** Main variables and processes of the example model and its extension.

## References


Bondeau, A., Smith, P. C., Zaehle, S., Schaphoff, S., Lucht, W., Cramer, W., Gerten, D., Lotze-Campen, H., Müller, C., Reichstein, M., et al.: Modelling the role of agriculture for the 20th century global terrestrial carbon balance, Global Change Biology, 13, 679–706, 2007.

CF Standard Names, 2018: CF Standard Names, http://cfconventions.org/standard-names.html, http://cfconventions.org/standard-names.html, accessed on 2018/09/18.

Fischer-Kowalski, M.: On the Childhood and Adolescence of a Rising Conceptual Star, in: The International Handbook of Environmental Sociology, Edward Elgar Publishing, Cheltenham, UK, 1997.

Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python, http://www.scipy.org/, [Online; accessed 2015-05-30], 2001.

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M. J., Terrel, A. R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., and Scopatz, A.: SymPy: symbolic computing in Python, PeerJ Computer Science, 3, e103, https://doi.org/10.7717/peerj-cs.103, 2017.

Nitzbon, J., Heitzig, J., and Parlitz, U.: Sustainability, collapse and oscillations of global climate, population and economy in a simple World-Earth model, Environmental Research Letters, 12, 074 020, https://doi.org/10.1088/1748-9326/aa7581, 2017.

Otto, I. M., Wiedermann, M., Cremades, R., Donges, J. F., Auer, C., and Lucht, W.: Human agency in the Anthropocene, Ecological Economics, 167, 106 463, 2020.

Robinson, D. T., Di Vittorio, A., Alexander, P., Arneth, A., Michael Barton, C., Brown, D. G., Kettner, A., Lemmen, C., O'Neill, B. C., Janssen, M., Pugh, T. A., Rabin, S. S., Rounsevell, M., Syvitski, J. P., Ullah, I., and Verburg, P. H.: Modelling feedbacks between human and natural processes in the land system, Earth System Dynamics, 9, 895–914, https://doi.org/10.5194/esd-9-895-2018, https://www.earth-syst-dynam-discuss.net/esd-2017-68/, 2018.

Syvitski, J. P. M., Hutton, E. W. H., Piper, M. D., Overeem, I., Kettner, A. J., and Peckham, S. D.: Plug and Play Component Modeling – The CSDMS2.0 Approach, in: International Environmeltal Modelling and Software Society (iEMSs) 7th Intl. Congress on Env. Modelling and Software, San Diego, CA, USA, 2014.

van Vuuren, D. P., Kok, M., Lucas, P. L., Prins, A. G., Alkemade, R., van den Berg, M., Bouwman, L., van der Esch, S., Jeuken, M., Kram, T., et al.: Pathways to achieve a set of ambitious global sustainability objectives by 2050: explorations using the IMAGE integrated assessment model, Technological Forecasting and Social Change, 98, 303–323, 2015.

Wikipedia: Wikipedia article on "Society" (last checked on 12-23-2017), https://en.wikipedia.org/wiki/Society, 2017.

Williamson, O. E.: Transaction cost economics: how it works; where it is headed, De economist, 146, 23–58, 1998.

World Bank CETS codes, 2017: World Bank CETS codes, https://datahelpdesk.worldbank.org/knowledgebase/articles/201175-how-does-the-world-bank-code-its-indicators, https://datahelpdesk.worldbank.org/knowledgebase/articles/201175-how-does-the-world-bank-code-its-indicators, accessed on 2018/09/18.