

# FLUME: The Flexible Unified Model Environment

EGU General Assembly

April 2006

A. Treshansky, S. Mullerworth, M. Carter, & M. Christoforou

# 1. Introduction



The FLUME project will provide a new software infrastructure for Earth System Modelling at the Met Office. It will separate infrastructure code from scientific code and redesign both so that they are able to be more freely combined with each other and with externally provided codes. Scientific code will be modularised and coupled through *auto-generated* code.

An advantage of this approach is that it can be extended to work with different coupling frameworks. Using techniques derived from the Bespoke Framework Generator (BFG) developed at the University of Manchester, FLUME models will be able to couple using a simple argument-passing interface or a put/get interfaces such as that defined by PRISM.

In order to facilitate automatic code-generation, FLUME models and jobs will be described using FLUME Metadata.

Using FLUME effectively *insulates* a scientist from infrastructure details.

## 2. FLUME Models

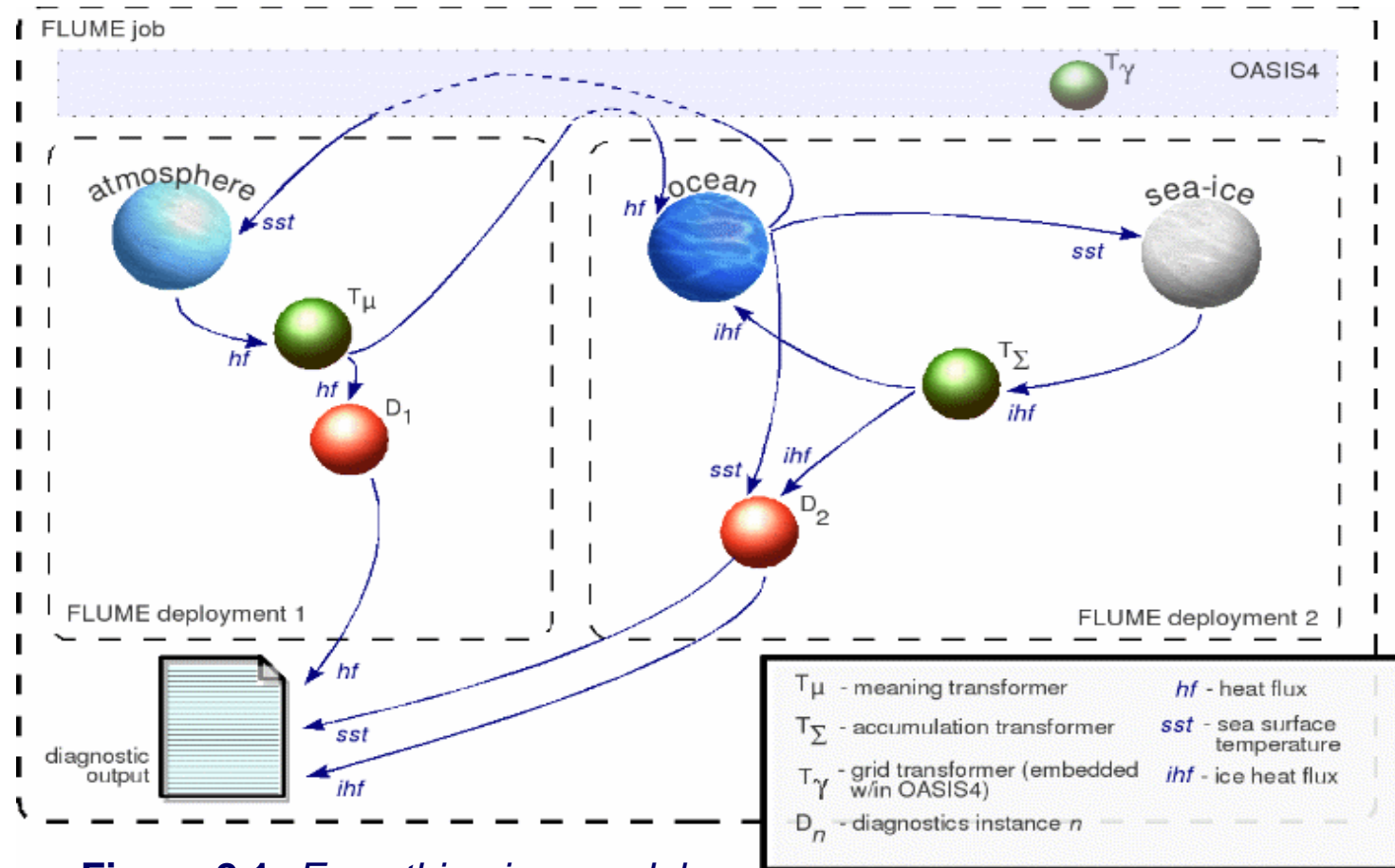


In order to be fully 'FLUME conformant':

- a model, including all of its unique inputs & outputs, must be defined in FLUME Metadata;
- a model's entry points must be implemented as a set of subroutines. Each subroutine may be categorised as an 'init', 'run', 'final', 'checkpoint', or 'restart' subroutine. A single model timestep may be made up of calls to several distinct (run) subroutines;
- all data required & provided by a model must be known by FLUME and made available, either as arguments of the model's subroutines or via the put/get API;
- a model should not terminate itself; if an error occurs it should inform FLUME.

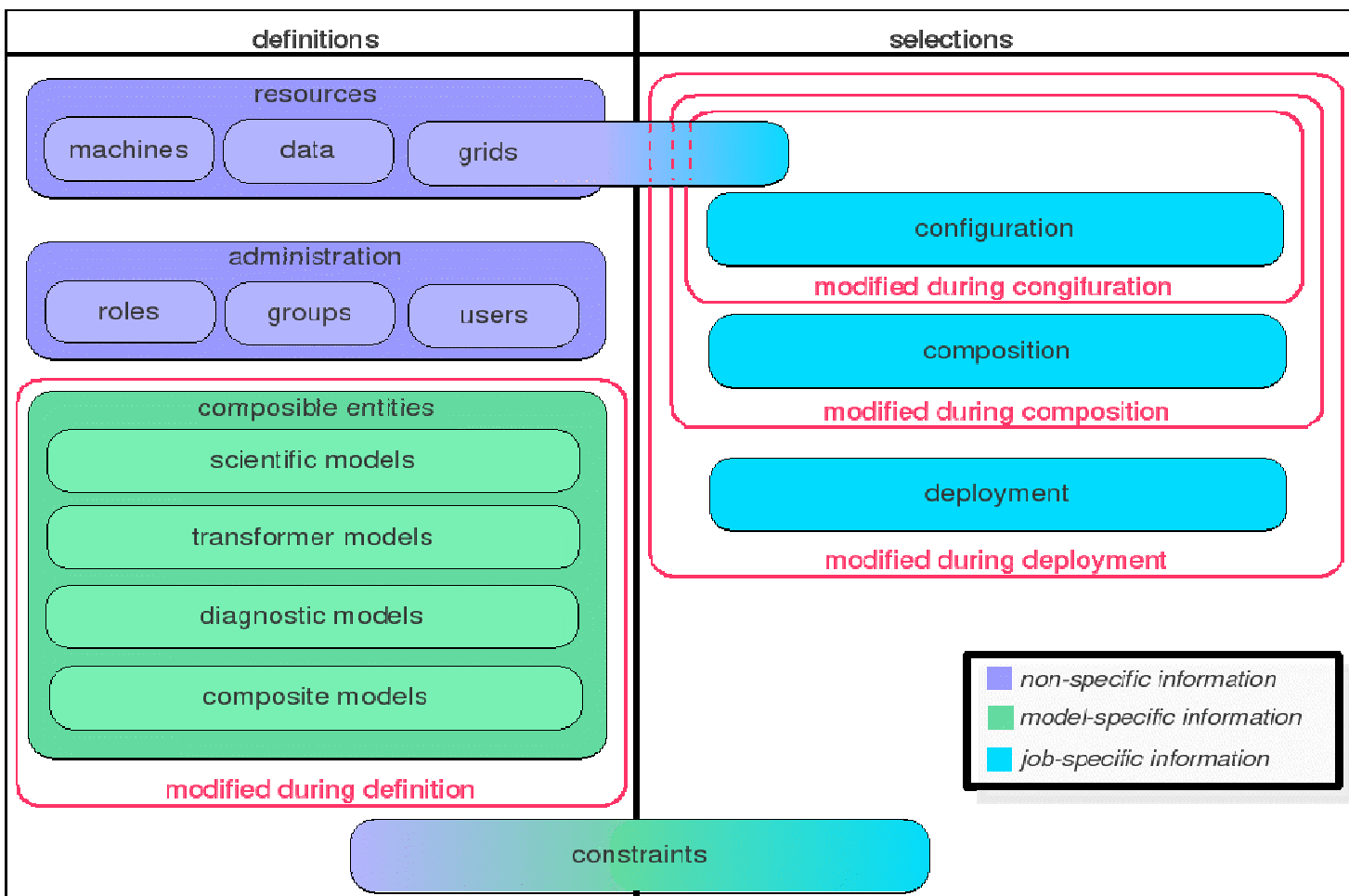
# 2. FLUME Models (cont.)

FLUME treats each component of a job the same. Hence, in **Figure 2.1** FLUME doesn't care that internally some models are doing science, others are transforming data, and others are collating diagnostics. It only cares about the items in blue: the fields and the arrows connecting them. This is different from the 'built-in' transformations that can occur as part of the PRISM coupler.



**Figure 2.1:** *Everything is a model*

# 3. FLUME Metadata



Every aspect of a FLUME job should be described by FLUME Metadata.

Figure 3.1 shows some different ways of categorising metadata. Some of these categories are highlighted below.

Figure 3.1: Categories of metadata

# 3. FLUME Metadata (cont.)



Composable entities (CEs) include anything whose data can form part of a FLUME composition, such as scientific models or transformers. The CE schema describes all possible subroutines, data values, and coupling options the model supports.

A configuration is a specific instance of a CE; it encodes a user's choices.

A composition describes the set of connections amongst the data provided and required by each subroutine in each model of a job.

A deployment describes how a job should be run on specific computing resources. It includes how each model's data should be partitioned, how (or if) the entire job should be parallelised, and how the subroutine calls forming the control code should be scheduled.

Constraints are embedded within FLUME metadata. They are used to enforce logical relationships among metadata elements. All constraints must be satisfied for FLUME Metadata to be valid.

# 3. FLUME Metadata (cont.)

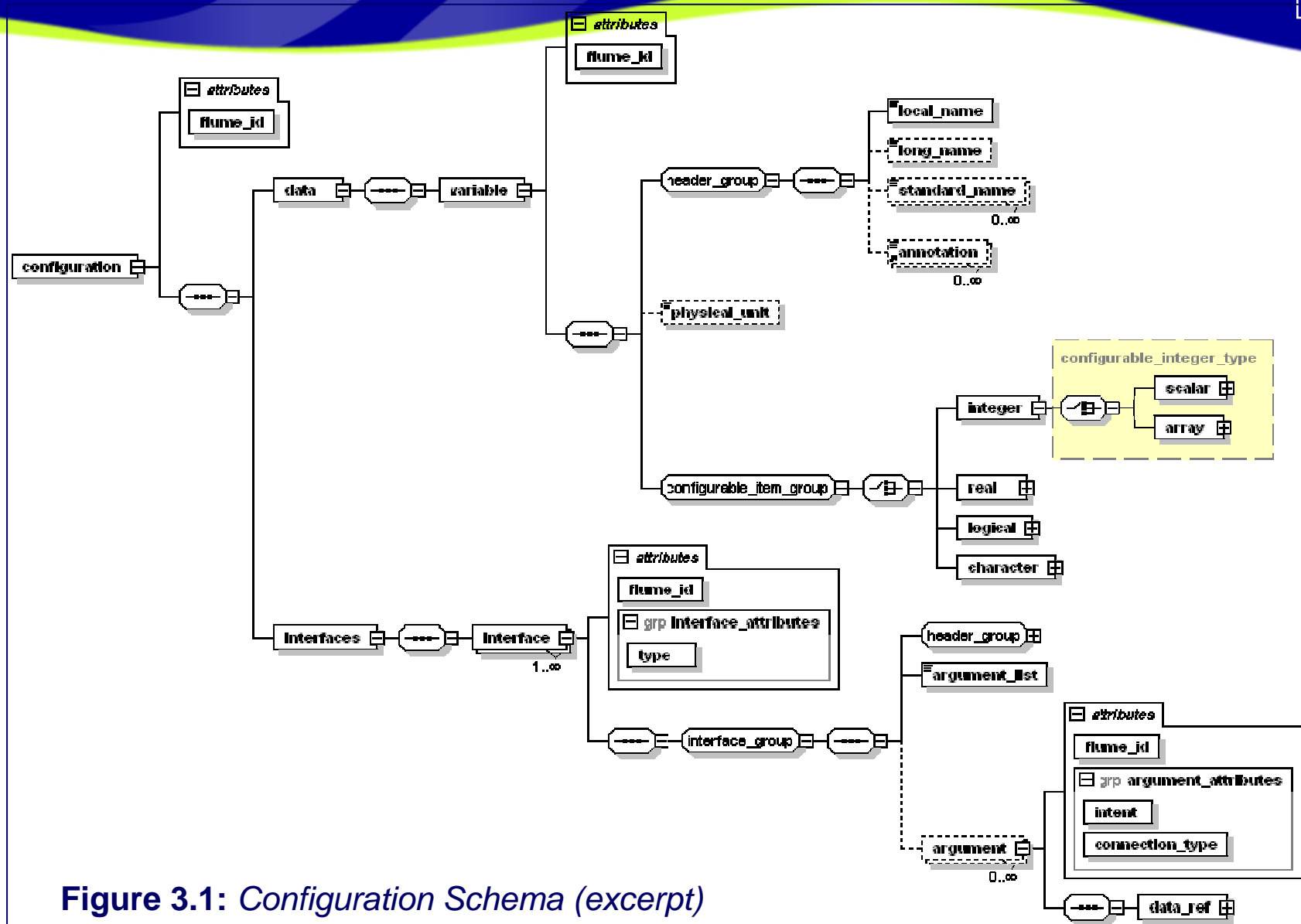


Figure 3.1: Configuration Schema (excerpt)

# 3. FLUME Metadata (cont.)

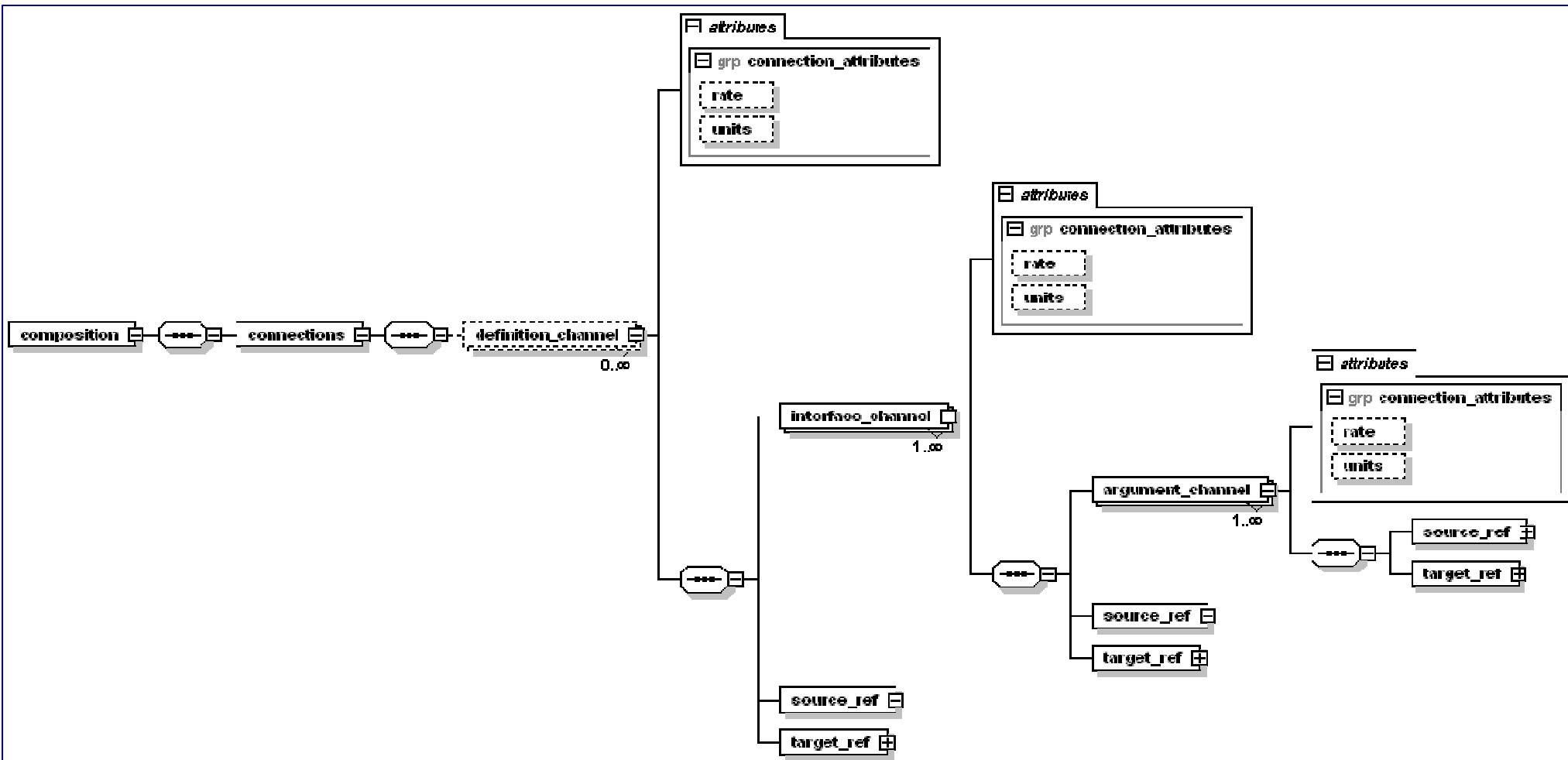


Figure 3.2: Composition Schema (excerpt)

# 3. FLUME Metadata (cont.)

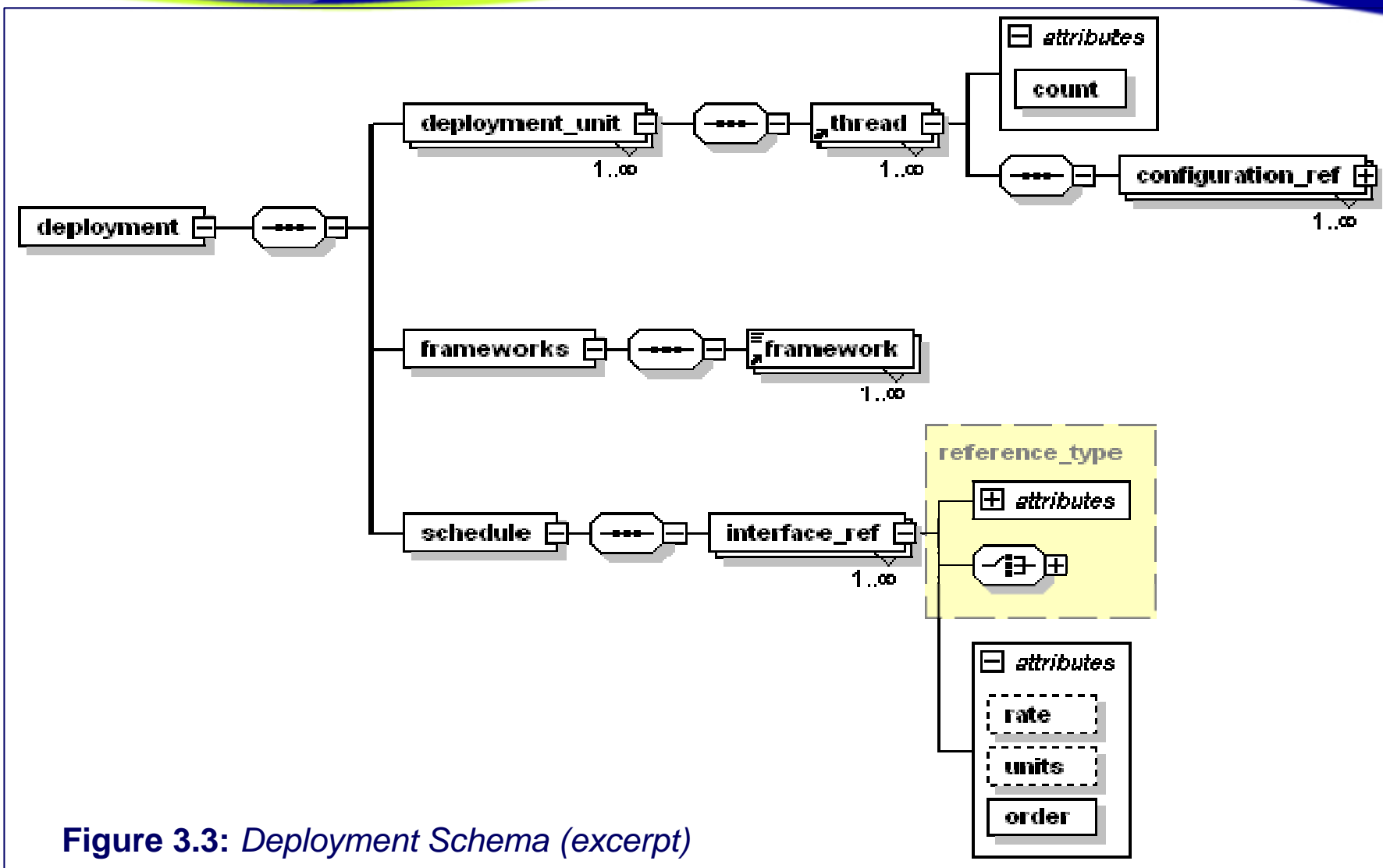


Figure 3.3: Deployment Schema (excerpt)

# 4. FLUME Prototype

The FLUME prototype is a library containing minimal infrastructure code with 'adapters' for other systems such as the OASIS4 coupler, MPI, standard grid descriptors, etc. The prototype allows FLUME to manage state, couple data, partition models, and handle errors.

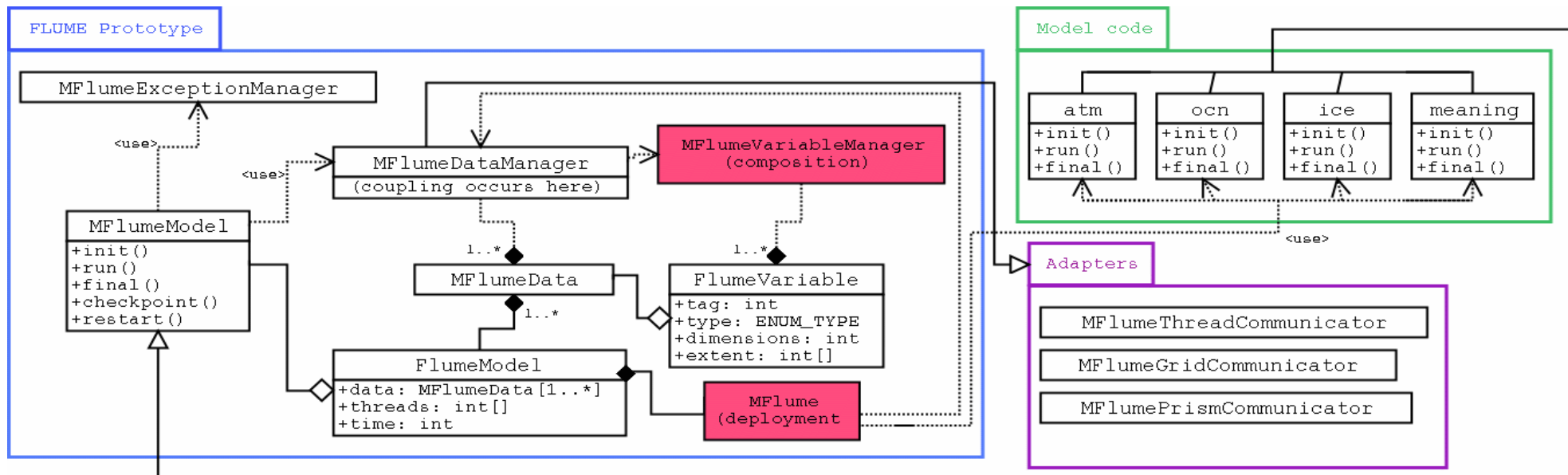


Figure 4.1: The FLUME prototype

The prototype provides code to use with FLUME auto-generated code. Model code should not need to use the prototype – in fact to do so would violate FLUME's ideal of separating science & infrastructure code. The red boxes in **Figure 4.1** show how auto-generated code incorporates models into the prototype.

# 5. FLUME Auto-Generation



The process of creating a FLUME job is as follows:

- The UI presents options to the user based on definitions metadata.
- The user selects from these options creating configuration, composition, & deployment metadata through the UI.
- The user validates and saves the set of metadata as a FLUME job.
- The user passes that job to the FLUME code-generator which produces the code to bind the components together.
  - A. The code-generator is modularised into multiple XSL transformation phases, each one building on the XML output of the previous phase.

# 5. FLUME Auto-Generation (cont.)



```
module MFlume

real, dimension(:,:,,:), save :: sst, ihf, hf

subroutine run()

  use ocn, ocn_init=>init, ocn_run=>run
  use ice, ice_init=>init, ice_run=>run
  use atm ! just needed for tags

  type(FlumeModel) :: ocnModel, iceModel

  call MFlumeModel_init(ocnModel)
  call MFlumeModel_init(iceModel)
  ! register put/get couplings..
  call MFlumeDataManager_couple(ocnModel,TAG_sst, &
    atmModel,TAG_sst)

  do i=1,NTS
    if(mod(i,ocn_freq).eq.0)then
      call ocn_run(hf,ihf,sst)
      if(mod(i,ocn_coupling_freq).eq.0)then
        call put(TAG_sst,sst)
      endif
    endif
    if(mod(i,ice_freq).eq.0)then
      call ice_run(ihf,sst)
    endif
  enddo

end subroutine run
```

**Figure 5.1** shows some auto-generated code. Note that put/get couplings have to be registered with the FLUME layer – this ensures that the call to put routes sst appropriately. In contrast, argument-passing coupling simply occurs by using the same local variable names.

**Figure 5.1:** Auto-generated control code (excerpt)