

**PRISM**  
**System Specification**  
**Handbook**

**Version 1.0**

Edited by:

Eric Guilyardi, Reinhard Budich, Guy Brasseur and Gerbrand Komen



**PRISM**  
**System Specification**  
**Handbook**

**Version 1.0**



# **PRISM**

## System Specification

Version 1.0

Edited by

Eric Guilyardi

Centre for Global Atmospheric Modelling; University of Reading, United Kingdom

Reinhard G. Budich

Max Planck Institut für Meteorologie, Hamburg, Germany

Guy P. Brasseur

Max Planck Institut für Meteorologie, Hamburg, Germany

Gerbrand Komen

Het Koninklijk Nederlands Meteorologisch Instituut, The Netherlands

February 2003

ISBN Nr. 90-369-2217-8

This handbook in its most recent edition can be found under <http://prism.enes.org>

It can be ordered via email from [prism-director@enes.org](mailto:prism-director@enes.org) .

## Contributors

*MICK CARTER*

Hadley Centre for Climate Prediction and Research, United Kingdom

*GABRIELLA DEMARTINO*

Koninklijk Nederlands Meteorologisch Instituut, Netherlands

*RALF DÖSCHER*

Rosby Centre, Sveriges meteorologiska och hydrologiska institut, Sweden

*HELGE DRANGE*

National Energy Research Scientific Computing Center, Norway

*THIERRY FICHEFET*

Université Catholique de Louvain-la-Neuve, Belgium

*MARIE-ALICE FOIJOLS*

Institut Pierre Simon Laplace, France

*ERIC GUILYARDI*

Centre for Global Atmospheric Modelling University of Reading, United Kingdom

*ROSALYN HATCHER*

Hadley Centre for Climate Prediction and Research, United Kingdom

*LUIS KORNBLUEH*

Max Planck Institut für Meteorologie, Germany

*CLAES LARSON*

European Centre for Medium-Range Weather Forecasts, International

*STEFANIE LEGUTKE*

Max Planck Institut für Meteorologie, Germany

*CORINNE LEQUERÉ*

Max Planck Institut für Biogeochemie, Germany

*ANGELO MANGILI*

Centro Svizzero di Calcolo Scientifico, Eidgenössische Technische Hochschule Zürich, Switzerland

*SERGE PLANTON*

Météo-France, France

*JAN POLCHER*

Institute Pierre Simon LaPlace, France

*RENÉ REDLER*

C&C Reserch Laboratories NEC Europe Ltd., Germany

*MARKKU RUMMUKAINEN*

Rosby Centre, Sveriges meteorologiska och hydrologiska institut, Sweden

*MARTIN STENDEL*

Danmarks Meteorologiske Institut, Denmark

*HANNES THIEMANN*

Modelle & Daten, Max PlanckInstitut für Meteorologie, Germany

*SOPHIE VALCKE*

Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, France

*NILS WEDI*

European Centre for Medium-Range Weather Forecasts, International





## *Foreword to the PRISM System Specification*

*Guy Pierre Brasseur and Gerbrand Komen*

*PRISM and ENES co-coordinators*

A major challenge for the climate research community is the development of comprehensive Earth system models capable of simulating natural climate variability and human-induced climate changes. Such models need to account for detailed processes occurring in the atmosphere, the ocean and on the continents, including physical, chemical and biological processes on a variety of spatial and temporal scales. They have also to capture complex nonlinear interactions between the different components of the Earth system and assess how these interactions can be perturbed as a result of human activities.

Accurate scientific information is required by government and industry to make appropriate decisions regarding our global environment, with direct consequences on the economy and lifestyles. It is therefore the responsibility of the scientific community to accelerate progress towards a better understanding of the processes governing the Earth system and towards the development of an improved predictive capability. An important task is to develop an advanced software and hardware environment in Europe, under which the most advanced high resolution climate models can be developed, improved, and integrated. To achieve this, over 30 institutions, including university departments, research centres, meteorological services, computer centres and industrial partners are collaborating in a European Network for Earth System Modelling (ENES, <http://www.enes.org>).

One of the activities fostered by ENES is the European Union's **PRISM infrastructure project**, which will provide a flexible environment to easily assemble and run Earth System Models (<http://prism.enes.org>). The present handbook, produced by the PRISM System Specification Work group under the chairmanship of Eric Guilyardi, is the result of the first phase of the project. The work group was appointed by the PRISM Steering Committee in January 2002 to ensure proper gathering of inputs and requirements from the scientific and technical communities. It surveyed existing solutions, tools, and associated expertise the project can benefit from. The handbook provides the basis for further implementation of the PRISM infrastructure. In particular, the PRISM model assembly work package will now collect and integrate the component models and the PRISM software packages as they become available.

We expect the handbook to be a living document, that will stimulate discussion about standards in Earth System modelling, not only in Europe but also in research communities elsewhere in the world. The PRISM System Specification Work group will continue to be active and welcomes comments ([ericg@met.reading.ac.uk](mailto:ericg@met.reading.ac.uk)).

We acknowledge the hard work and dedication of the PRISM System Specification Work group and PRISM Director Reinhard Budich in compiling this volume.



# Content

<b>CONTRIBUTORS</b> .....	<b>7</b>
<b>FOREWORD TO THE PRISM SYSTEM SPECIFICATION</b> .....	<b>9</b>
<b>CONTENT</b> .....	<b>11</b>
<b>EXECUTIVE SUMMARY</b> .....	<b>15</b>
<b>REDOC</b> .....	<b>19</b>
<b>I - SCIENTIFIC AND FUNCTIONAL REQUIREMENTS</b> .....	<b>19</b>
<b>I.1 Scientific Aspects</b> .....	<b>19</b>
I.1.1 PRISM components .....	19
I.1.2 Conservation principles.....	19
I.1.3 Universal parameters.....	20
I.1.4 General Recommendations.....	21
<b>I.2 Inputs to Components and Physical Interfaces</b> .....	<b>22</b>
I.2.1 Introduction.....	22
I.2.2 Discussion .....	23
<b>I.3 Simulations and Coupling Configuration</b> .....	<b>25</b>
I.3.1 List of Simulations .....	25
I.3.2 Coupler Requirements.....	26
<b>I.4 End User Interface</b> .....	<b>35</b>
I.4.1 Functional Requirements.....	35
I.4.2 Operational Requirements.....	42
<b>II - DESIGN OPTIONS FOR THE PRISM ARCHITECTURE</b> .....	<b>43</b>
<b>II.1 PRISM System General Architecture</b> .....	<b>43</b>
II.1.1 Terminology and Concepts .....	43
II.1.2 System model.....	46
II.1.3 Software life cycle.....	50
II.1.4 Security .....	51
II.1.5 Specific Design Choices and Constraints .....	53
II.1.6 Software Packages Investigated .....	53
II.1.7 Discussion on the architectural choices.....	55
II.1.8 Cooperating systems .....	56
II.1.9 Risks.....	57
<b>II.2 PRISM Coupler, Including Coupler Review</b> .....	<b>59</b>
II.2.1 Introduction - Definitions.....	59
II.2.2 Possible Requirements and Design Options for the PRISM Coupler .....	62
II.2.3 Coupler review .....	73
<b>II.3 Data Management, Analysis, Visualization and Archiving</b> .....	<b>79</b>
II.3.1 Processing Library .....	79
II.3.2 Visualisation.....	79
II.3.3 High Level Architecture.....	80
<b>II.4 PRISM user interface</b> .....	<b>83</b>
II.4.1 Design Options and Constraints .....	83
II.4.2 Commercial Components.....	84
II.4.3 Hardware/Platform Interfaces .....	84
II.4.4 Software Interfaces.....	84
II.4.5 Communication Interfaces .....	84
II.4.6 Performance .....	84
II.4.7 Extensibility .....	85
<b>III - PRISM SYSTEM CONSTRAINTS</b> .....	<b>87</b>
<b>III.1 Review of Component Models</b> .....	<b>87</b>
III.1.1 List of Models .....	87

III.1.2	Technical Aspects .....	88
III.2	<i>Review of Existing and Future Target Computer Platforms</i> .....	91
III.2.1	Evolution of the HPC Market.....	91
III.2.2	HPC Architectures: State-of-the-Art and Trends.....	92
III.2.3	Conclusion and Recommendations .....	97
III.3	<i>Software Engineering Process, Coding Rules and Quality Standard</i> .....	99
<b>ARCDI</b> .....		<b>103</b>
<b>I -</b>	<b>BASIC CHOICES</b> .....	<b>103</b>
I.1.1	PRISM Components.....	103
I.1.2	Conservation Principles.....	103
I.1.3	Universal parameters.....	103
<b>II -</b>	<b>A PROPOSAL FOR STANDARD ATMOSPHERE/OCEAN/SEA ICE INTERFACES</b> .....	<b>105</b>
II.1	<i>Introduction</i> .....	105
II.2	<i>Interface design constraints</i> .....	105
II.2.1	General constraints.....	105
II.2.2	Model components constraints.....	106
II.3	<i>The Proposal</i> .....	107
II.3.1	Introducing new modules .....	108
II.3.2	Physical interfaces.....	109
II.3.3	Discussion.....	110
<b>III -</b>	<b>SYSTEM ARCHITECTURE</b> .....	<b>111</b>
III.1	<i>Introduction</i> .....	111
III.2	<i>Terminology and Concepts</i> .....	112
III.3	<i>The Local System, a Basis for Development</i> .....	113
III.4	<i>Architecture of the Generalized Web Services Concept</i> .....	113
III.5	<i>From Analysis to Design: The Process-to-Component-Translation of the PRISM System</i> .....	116
III.6	<i>Proposed Architecture</i> .....	124
III.7	<i>Specifications, Definitions and Standardized Interfaces Needed for the Implementation of the System</i> .....	127
III.8	<i>From Design to Implementation: The Component-to-Deployment Translation</i> .....	128
III.9	<i>Implementation Plan, Deliverables and Desirables</i> .....	130
III.10	<i>Integration with other Projects</i> .....	133
III.11	<i>Risks</i> .....	134
<b>IV -</b>	<b>SYSTEM COMPONENTS</b> .....	<b>135</b>
IV.1	<i>PRISM Coupler and I/O Library</i> .....	135
IV.1.1	Coupled Model High Level Architecture .....	135
IV.1.2	Detailed Functionalities for the PRISM Coupler and I/O Library.....	140
IV.2	<i>Diagnostics and Visualization</i> .....	153
IV.2.1	Introduction.....	153
IV.2.2	Meta-Data and File Format Definitions for Model Output and Data Exchange .....	153
IV.2.3	High-level Architecture.....	154
IV.2.4	Processing Library .....	155
IV.2.5	Visualization .....	157
IV.2.6	Archiving .....	160
IV.2.7	Statement of licensing .....	160
IV.3	<i>User Interfaces</i> .....	161
IV.3.1	Introduction.....	161
IV.3.2	General Implementation.....	161
IV.3.3	Administration Interface .....	162
IV.3.4	Adaptations .....	162
IV.3.5	New features .....	164
<b>V -</b>	<b>SOFTWARE ENGINEERING PROCESS, CODING RULES, AND QUALITY STANDARDS</b> .....	<b>165</b>
V.1	<i>Introduction</i> .....	165
V.2	<i>General Software Development Guidelines</i> .....	166
V.3	<i>Coding Conventions for the Development of PRISM Components</i> .....	167
V.4	<i>Component and Unit Testing</i> .....	182
V.5	<i>System Testing and Validation</i> .....	184
V.6	<i>Code Maintenance Issues</i> .....	186

**REFERENCES ..... 191**

**GLOSSARY ..... 193**

**APPENDICES: ..... 201**

**APPENDICES TO REDOC I.2.: ..... 202**

**APPENDICES TO REDOC I.2.: ..... 202**

**APPENDIX TO REDOC I.3: ..... 226**

**TABLES ..... 235**

**FIGURES ..... 237**



## *Executive Summary*

This document presents the first version of the PRISM System Specification Handbook. It surveys the existing solutions, tools, and associated expertise the project can benefit from. The document consists of two parts. The first part, "Requirements, Design Options, and Constraints" (REDOC) summarizes the requirements on a coupling system that is regarded as 'ideal' in the sense that it fits the perceived long-term (order 5/10 years) requirements of the scientific community undertaking earth-system simulations, both from a scientific point of view as well as from the users point of view. It gives in addition relevant design options as well as foreseeable constraints by models and technologies. The second part, "Architecture Choices, Detailed Design, and Implementation" (ARCDI), describes the actual choices that will guide the work envisaged for the 3 years of the project. Due to the numerous interactions between the groups, several issues are not yet resolved or even foreseen and will require more inputs from the community. Hence this first version of the handbook (and in particular ARCDI) should be understood as evolving working documents.

**Science:** Subsystems of the PRISM earth system model are: atmosphere, atmospheric chemistry, land surface, ocean, ocean biogeochemistry, and sea ice. These components can be either global or regional. The main scientific principles that will be followed are: physically based local conservation of fluxes across coupling interfaces and global conservation of heat, water, ocean salt content, as well as (biogeo-) chemical species. Parameters and functions common to more than one component model will also be consistently defined (i.e., earth radius,  $\pi$ , gravity...).

Standard PRISM physical and algorithmic interfaces between components are proposed, inviting the modelling teams both to comment them and to organize their developments to adopt them over time. Though clear guidance will be given for the implementation, it is not expected that all models will comply with the proposed physical interfaces within the first three years of the project. Thus, the initial assembly and testing of the PRISM system will use non-standard physical interfaces. This possibility will remain an option in the final PRISM system. Particular treatment of atmosphere / atmospheric chemistry and ocean / ocean biogeochemistry interfaces is needed since the models share the same physical space and hence many physical processes.

As of Sept 2002 about 30 models (global and regional) have expressed interest to be part of the PRISM system:

*Atmosphere:* ARPEGE-Climat, ECHAM5, HIRHAM, LMDZ, RACMO, RCA, Unified Model atmosphere, HadRM3H;

*Atmospheric chemistry:* INCA, KNMI\_TM, MOZART;

*Land surface:* ISBA, MOSES, ORCHIDEE, RCA-soil;

*Ocean:* C-HOPE, MICOM, MOM, HYCOM, OPA, RCO, Unified Model Ocean;

*Sea ice:* LIM, NERSC, RCI, UMI;

*Ocean biogeochemistry.* HADOCC, PISCES, HAMOCC.

**Technical development:** The core of the PRISM technical developments consists of the system architecture, the PRISM coupler and I/O library, the diagnostic output library and the user interface.

The PRISM system architecture will deliver the infrastructure necessary to configure, submit and monitor coupled experiments. The design will allow these activities to be done remotely, via the

web. A local system will be provided to enable model developers to run the configuration tool without the web service infrastructure to test their models. The local system is based on the same software to ensure a transparent transition between the local and the web services infrastructure system. A centralized architecture that minimizes the administration and duplication of resources will best fulfil the PRISM requirements (remote access, modularity, extendibility). The system proposed will meet the specific demands of the three types of actors within PRISM: the users, the model developers and the PRISM system administrators. The PRISM system architecture will benefit from proven software packages and expertise available within the community (PrepIFS, SMS,...).

The PRISM coupler will drive the whole coupled model, ensuring the synchronization of the different component models and the exchange of the coupling fields directly between the components or via additional coupling processes. When needed, the coupler will perform transformations on the exchange fields. Another important part of the coupler is a library linked to each component model by which they are interfaced to the rest of the PRISM system. As I/O and data exchange share many characteristics, it was decided to develop one common model library for both purposes. The different constituents of the PRISM coupling system are therefore the Driver, the Transformer, and the PRISM System Model Interface Library (PSMILe). The PSMILe includes the Data Exchange Library, which performs the exchanges of coupling data, the I/O library, coherence checks of metadata and data, and local transformation routines. This PRISM coupler will benefit from the expertise gathered with the OASIS coupler, developed at CERFACS and used extensively by the community.

For diagnostic outputs, a flexible library of tools will be built to facilitate processing and analysis of data in the common PRISM data format and to promote sharing of data and analysis programs. High-End and Low-End graphical interfaces, respectively local and remote, will be developed to process and display the data. The meta-data standard chosen for the output of PRISM model data and for data exchange is the CF convention. NetCDF will be the supported file format for output and data exchange but the system design will allow for other file formats to be added. NetCDF-CF is emerging as the international standard for earth system and climate data and its use is encouraged in the PRISM community.

A PRISM Software Developer's Guide specifies guidelines, conventions, and standards for design, implementation, documentation, and quality assessment of any software that will be developed in the frame of the PRISM project under the consideration of portability, sustained performance, and ease of use.



Requirements,  
Design Options,  
Constraints:

REDOC



# REDOC

## I - Scientific and Functional Requirements

### I.1 Scientific Aspects

*The PRISM model components and their general relations are defined, and scientific principles, which need to be obeyed for any combination of components, are presented. Global parameters that need to be consistently defined in all components are discussed.*

---

#### I.1.1 PRISM components

Understanding the climate system is one of the great challenges at the beginning of the millennium. Coupled Climate Models (CCMs) are the most comprehensive tools to study the climate system and the mechanisms through which the different components interact. However, these models can be extremely complex, therefore care must be given to the technical aspects of the CPU intensive simulations based on such CCMs, as well as on the physical interactions between their components.

In the framework of the PRISM project, an attempt is undertaken to bring models of subsystems of the climate into a common framework. The following components are involved in PRISM: atmospheric general circulation models (AGCM), atmospheric chemistry models (AC), ocean general circulation models (OGCM), models of the ocean biogeochemistry (OC), land surface (LS) and sea ice (SI) models. The component models can be global or regional. The envisaged modularity of the PRISM system would allow to include additional components such as ocean waves, continental ice sheets, and volcanic or solar models.

#### I.1.2 Conservation principles

Field exchange between the PRISM components should meet all conservation laws, and unphysical internal sinks in the coupled model should not exist. This is a prerequisite for the coupled model to be able to run stable simulations for indefinitely long time periods (e.g. paleo simulations). Ideally, all conservation laws have to be satisfied locally up to the accuracy of the algorithms employed in the models. A scheme that is locally conservative necessarily guarantees the positive definiteness of non-negative quantities. This would imply global conservation as well, provided that all fluxes are consistent across interfaces, or in other words, that sinks in one model component have a corresponding source in the component at the other side of the interface. The exchange of any specific quantity between component models should be accompanied by the exchange of other quantities associated to it in the source model and subject to a conservation law. If for example the formation of snow in a particular atmospheric model goes along with the release of latent heat of fusion, the same amount of heat of fusion must enter the ocean when the snow falls into the ocean. Likewise, if raindrops do not change temperature in the atmosphere during their lifetime after formation, then no sensible heat flux should occur with the flux of precipitated water. Accordingly, conservation laws can be formulated for continental or glacier/ice sheet runoff. Conservation

laws are also required for biogeochemical elements (i.e. C, P, N, S, Si, Fe, ...) and isotopes ( $^{14}\text{C}$ ,  $^{13}\text{C}$ ,  $^{17}\text{O}$ ,  $^{18}\text{O}$ ,  $^{15}\text{N}$ , ...).

### I.1.3 Universal parameters

To allow for a meaningful communication between the components of PRISM, additional prerequisites need to be met. In particular, universal parameters (i.e. parameters and equations common to more than one PRISM component) need to be consistently defined. The following list gives some examples:

Astronomical parameters	Earth radius Calendar used (this determines length of day and year and angular velocity)
Physical parameters	Gravity acceleration Solar constant and its variability Latent heat of fusion/evaporation for ice/water Density of pure/sea water Specific heat of pure/sea water Reference density of sea water Stefan-Boltzmann constant Full equation of state for sea water (density as a function of temperature, salinity and pressure)
Model parameters	Initial and stop date of experiment/run Length of integration Frequency of saving restart files Frequency of saving analysis files

Table 1: Examples of Universal Parameters

Of particular importance is the choice of the model calendar, which is essential to synchronise and control the flow of the PRISM system and to trigger events of the PRISM components. Models use calendars with a constant number of days per year (30 days per month or 365 days per year ignoring leap years) or the standard Gregorian calendar with 365 or 366 days per year. Other calendars with orbital characteristics different from the present ones of the earth (e.g. different lengths of days) may be needed for paleo simulations or simulations of other planets. While the Gregorian calendar should be preferred, it is essential that the other options are not excluded in the PRISM specifications. However, the same calendar must be used by each component in a coupled simulation.

Several calendar/time-control tools exist, and three of them are currently under investigation, namely

- The ECHAM5 tool developed at MPI-HH,
- The ESMF tool developed at NCAR, and
- The IOIPSL module.

A summary of functionalities as well as comments on the ease of use of the tools and a recommendation for use will be distributed in the near future.

## I.1.4 General Recommendations

Other points raised during the discussions of the by PRISM system specifications:

- Atmosphere models are sometimes coupled to simplified versions of other components of the climate system (e.g. soil water and energy sub-component, simplified chemistry, mixed-layer ocean, thermodynamic sea ice, ...). The PRISM system should be able to preserve fast communication with these simplified components.
- From a physical point of view, sea ice and the upper ocean are intimately linked (sea ice is the solid phase of sea water), and sea-ice growth, decay, and drift mutually affect thermodynamics and dynamics of the oceanic mixed layer. Treating them independently is considered as problematic. If the sea ice - ocean system is to be separated into two components, it will be necessary to at least adhere to the following two requirements:
  - Sea ice and ocean grids should be congruent and the former should have a resolution at least equal to that of the latter. This is important in order to ensure conservation of freshwater, salt, heat and momentum both locally and globally. It is also crucial for appropriately resolving the ice front and horizontal freshwater fluxes at the interior of the ice pack.
  - Sea ice and ocean model time steps should be equal in order to guarantee that sea surface temperatures remain above the freezing point (or close to it if super-cooling is allowed) and that variations in mass, energy and momentum exchanges between the ice cover and the mixed layer occur on realistic time scales.
- The relation of RCMs to the global PRISM system components is rather straightforward. A global model component feeds into the corresponding component of an RCM, at the lateral, and in some cases, at the upper or lower boundaries. The relation can be one-way (GCM to RCM) or two-way (the RCM also returns information to the GCM). The latter is not planned during the PRISM project phase. Coupling may be online or offline. In online coupling, a GCM and a RCM are run concurrently, in offline calculations the GCM data are read from disk.

RCMs require large-scale forcing data at their lateral (geographical) boundaries in addition to the forcing data needed also by GCMs. When these data are provided by a GCM in an on-line coupled simulation, the RCM needs to lag the GCM sufficiently so that temporal interpolation between two successive lateral boundary data sets can be done. Interpolations in time and space have to be performed in order to obtain boundary data at the RCM's resolution and time-step.

## 1.2 Inputs to Components and Physical Interfaces

*At a first stage in the design of standard physical interfaces, an attempt was made to list all the inputs to the physical components identified in PRISM. A discussion of these initial lists follows. The second stage of this design is made in ARCDI with a proposal of standard physical interfaces.*

---

### 1.2.1 Introduction

The definition of a standard set of physical and algorithmic interfaces between the different PRISM component models is done in a two-stage process. First, each component model group (PRISM WP3b-h leaders and respective communities) was asked to list all inputs required by each component model to solve its prognostic/diagnostic equations. The list of inputs should be model independent and solely based on the modelled aspects of the physics of the earth system as we know it today, and on those aspects we anticipate to become important in future modeling studies. For each identified physical field the following was required:

1. An explanation of why this field is needed (corresponding process),
2. The SI unit
3. The shortest time scale that will presumably be physically resolved in future developments
4. Issues (like sub-grid scale, multiples options ...)
5. Rating (Essential, Desirable, May-be, Unlikely in the next 5-10 years)
6. Likely model component origin of field (Atmosphere, Atmospheric Chemistry, Land Surfaces, Ocean, Sea Ice, Ocean Biogeochemistry)

Work Package	Leader	Related documents / Appendices under <a href="http://prism.enes.org/">http://prism.enes.org/</a>
3b Atmosphere	Serge Planton	<a href="#">Results/Handbook/Appendices/redoc_1.2_App_atmos.html</a>
3c Atmospheric chemistry	Guy Brasseur	<a href="#">Results/Handbook/Appendices/redoc_1.2_App_atmoschim.html</a>
3d Land surfaces	Jan Polcher	<a href="#">Results/Handbook/Appendices/redoc_1.2_App_iss.html</a>
3e Ocean	Eric Guilyardi	<a href="#">Results/Handbook/Appendices/redoc_1.2_App_ocean.html</a>
3f Sea ice	Helge Drange	<a href="#">Results/Handbook/Appendices/redoc_1.2_App_seaice.html</a>
3g Ocean Biogeochemistry	Corinne Le Quéré	<a href="#">Results/Handbook/Appendices/redoc_1.2_App_oceanbio.html</a>
3h Regional models	Markku Rummukainen	<a href="#">Results/Handbook/Appendices/redoc_1.2_App_regional.html</a>

Table 2: Links to the input required for the standard physical and algorithmic interface

The results of the first round can be found at the sites given in Table 2.

Once these inputs were gathered, the definitions of the interfaces started. A couple of meetings and many email discussions were needed to launch this ambitious goal in the community. First results and analyses are presented below in the discussion section. This is an on-going process which will continue until all the groups involved agree on a common forward looking set of physical interfaces.

## 1.2.2 Discussion

A rapid inspection revealed that the interfaces need to be discussed again as they are too close to the current state of models and are not forward looking enough for an adoption within PRISM. Some of the critical points, which were noted are reported here.

Sub-grid scale surface heterogeneities of sea ice are very large at the resolution of presently used global models and it does not seem satisfactory to compute grid averaged fluxes over the coarse grid of an atmospheric model without any knowledge of the diversity of the surfaces. It thus seems foreseeable that there will be a need in the near future to account for these inhomogeneities in the computation of the fluxes at the atmosphere surface and therefore the physical interface should include the information needed to do this.

The approach used by the atmospheric group, to require the inputs separately from the various surfaces, is very forward looking. It opens a lot of doors for future development. In the same way the ocean interface should discriminate between the inputs on the sea-ice fractions and the open water in order to allow for parameterizations taking into account strongly localized processes (e.g. convection in leads).

Furthermore, one expects that the time scales at which these fluxes are computed become critical when the diurnal cycle needs to be taken into account and that there is a need for the sea-ice to be coupled to the planetary boundary layer of the atmosphere model. Thus the interfaces of the land surface and the sea-ice to the atmosphere should be very similar. As one may note in the table for the atmosphere, all surfaces are expected to provide different quantities. This means putting a heavy burden on the atmosphere models and is a potential impediment to future developments. It should be examined in detail why this is so and whether this difference can be based on physical principles or whether it is only due to some historical heritage. The distinction between the land surfaces and the ocean can be understood since their time scales of response to atmospheric forcing are different, but it is not obvious that there should be a distinction between land surface and sea-ice.

The difference in ocean and land-surface treatment raises the question whether we wish to cut ourselves off from the possibility of having a fast reacting ocean layer. It will need to be discussed in detail where the differences are and how they need to be treated as they bring a lot of complexity to those atmosphere models, which include both interfaces.

The atmospheric chemistry input table contains variables, which are from existing models and will probably evolve in the years to come. This will need to be factored into the proposed interface. The most prominent example for the evolution would be the distinction between convective and stratiform precipitation, which will disappear as cloud parameterizations become more complex. For the land surface it was chosen during the development of the PILPS-4c interface (Polcher et al. 1996) to request grid-box average and variance from the atmosphere (or another description of the second moment of the distribution) as they are more general and correspond to a general mathematical description of the sub-grid scale variability.

The proposed interfaces should avoid using model dependent variables. Examples for the latter type would be surface field capacity, temperature of the top soil layer, or vegetation type. These variables are conceptual in nature and can thus not be compared at our scales to observations and the range of values will vary greatly from one land-surface scheme to another. It does not seem very wise to base a general interface on these model dependent variables since they will change when for instance modifications will be made to the vertical discretisation or the representation of vegetation types. Their meaning will also change from one model to another as one model will for instance use biomes types for it's vegetation classes while the other will prefer plant functional types.

In many tables, surface temperature is used as a variable without any distinction being made between radiative surface temperature and the variable used in the turbulent exchange with the atmosphere. These two variables are of different nature and are linked to distinct physical processes. In some models they might be interchangeable but this is not generally true.

These issues need to be discussed again in detail and new proposals made (see ARCDI) before some final interfaces are adopted for PRISM.



## I.3 Simulations and Coupling Configuration

*In order to capture the scientific requirements of coupled simulations that should be feasible with the PRISM system both on the short term or at a later stage, desired and foreseen coupling configurations and associated coupler functionalities have been investigated by a questionnaire and ranked by the PRISM community. The evaluated questionnaire and comments can be found at <http://prism.enes.org>*

---

### I.3.1 List of Simulations

The PRISM system consists of a multitude of atmosphere, ocean, chemistry, land surface, sea ice and ocean biogeochemistry models, both global and for regional sub-domains (see list of models in ARCDI I.3).

These can be combined in different configurations. The simplest configuration of the PRISM system involves a coupled atmosphere/ocean model. Coupling of atmospheric and ocean models has been performed at a number of institutions for several years now. The PRISM system will allow for more complex configurations by adding modules for land surface, sea ice, atmospheric chemistry and oceanic biogeochemistry. Several combinations with different degrees of complexity are possible.

The inclusion of the other modules is a rather novel approach. Therefore a list of desirable simulations that should be feasible with the PRISM system both during the runtime of the project and in a longer perspective has been set up from the response of the modelling community to a questionnaire. The result is tabulated in appendix 'REDOC I.3 Requirement summary table'.

Regional coupled simulations proposed by the PRISM community for the runtime of the PRISM project are:

- Offline simulations driven by external dynamical fields which are read from files at given time intervals (e.g. GCM->RCM offline coupling)
- Interactive (online) simulation
- Regional chemical/transport model nested into a global chemical/transport model
- Run the regional simulation consisting of only a regional atmosphere and land surface model or alternatively a regional ocean and sea ice model. This means that the global model would provide the lateral boundary conditions for the atmosphere and the ocean, respectively, and the appropriate set of surface forcing over the regional domain.
- For regional models in PRISM, the aim should be to have the same flexibility of multiple coupling interfaces as in global model systems, i.e. concerning each of the atmosphere - ocean, atmosphere - land surface, land surface - ocean, atmosphere - sea ice and sea ice - ocean links
- It is necessary to run the regional simulation for any part of the globe

Simulations more likely to be done after the first PRISM project phase are:

- For regional models: simulations with same degree of complexity as in global simulations.

- For ocean biogeochemistry: offline simulations with degradation to coarser grid and coupled simulations with regional models.

Simulations, which are envisaged for a later phase, are:

- For regional models: two-way GCM-to-RCM coupled online AORCM simulations with the PRISM coupler.
- For ocean biogeochemistry: coupled simulations including fluxes of tracers associated with the hydrological cycle
- For sea ice: coupled simulations of several centuries duration.

### 1.3.2 Coupler Requirements

Clearly, not all requirements for the PRISM coupler can be met equally or at the same time. Therefore, a number of coupler options have been evaluated by the PRISM community to be either **Essential**, **Desirable**, **Maybe** of interest, **Useless** or that the person responding is **Not qualified** to answer. Comments and clarifying remarks from the PRISM community have been added. Note that in this context "essential" is judged from a scientific point of view, which does not imply that all of these points will be fulfilled during the project phase of PRISM in case the realisation is regarded as difficult.

#### *General Requirements*

*"The same version of a component model should be usable as part of a coupled model in the PRISM System and outside the PRISM system in stand-alone runs." - Essential*

The validation of the atmospheric component of a coupled model, the initialization phase of the coupled simulations, as well as many scientific applications (sensitivity experiments in complement to coupled simulations, 2xCO<sub>2</sub> simulations etc.) may require stand-alone simulation to be performed with the component models of the coupled model.

Regional models will be used both as a component in the PRISM system, but also elsewhere, driven with archived GCM results or global operational analyses. The interface should allow for both alternatives.

The opposite direction is also desirable: the PRISM system should provide pseudo models, which have the standard interface and read data from files.

E. g. atmospheric chemistry models need to be driven by atmospheric data without coupler using already existing data (files), e.g. from ECMWF.

*"The PRISM coupler allows intrinsic characteristics of the component models (e.g. length of a time step) to change at run-time". - Maybe*

This may be useful for periodic-synchronous coupling. Note, that in some atmospheric chemistry models the time step for certain processes differs from the advective time step, e.g. for aqueous chemistry. The time step of meteorological fields on input may differ from the advective time step as well.

*"The PRISM coupler allows coupling data characteristics of the component models (e.g. units, grid co-ordinates, mask, parallel decomposition, ...) to change at run-time". - Maybe*

This is of interest to the land surface component. Although difficult to control automatically, changes of land/sea mask values, e.g. from drying lakes under climate change conditions, may become an issue in the near future. Adaptive grids are unlikely to become an issue in ocean modelling in the next few years. There may also be interest in changing coupling data characteristics from a computational point of view: if parallel decomposition changes are foreseen as a means of dynamic load balancing.

For atmospheric chemistry modelling, we want to split up the atmospheric chemistry into 8 processes: coupling to advective transport, convective transport, turbulent transport, dry deposition, wet deposition, chemistry, emissions and photolysis. There should be a choice in the coupler configuration for any of these processes to be incorporated either in the atmospheric model or in the atmospheric chemistry model. The calling order and time step should be set in the coupler configuration for each of these processes separately. The parallel decomposition needs to be modifiable at runtime (to optimize in particular chemistry and photolysis).

## *Controller/Driver Requirements*

### **General:**

*"The PRISM System can also be used to assemble and run coupled models based on component models which do not conform to the PRISM Physical interfaces given that they include the well defined PRISM System Model Interface Library". - Essential*

PRISM will recommend a standard interface but the technical interface should allow both other interfaces and an evolution of the standard.

*"The PRISM System can be used to assemble and run coupled models based on an arbitrary number of component models (not only the full coupled model assembling all PRISM model components)." - Essential*

This modular approach is essential for climate science. Many climate simulations are (and will be in the future) performed using the atmospheric component without complete coupling. It may also be useful for atmospheric chemistry after PRISM (see previous paragraph).

### **Model Execution:**

*"The PRISM System should be able to run the different component models concurrently, in a regular sequence (one after the other), or in some pre-defined combination of these two modes." - Essential*

This covers the present mode of functioning of the coupled models and might be useful from the point of view of PRISM expandability and optimization on different platforms. Different alternatives should be possible. Running RCMs online with GCMs could imply concurrent execution, although slightly shifted in time (by a time step or longer if the GCM and RCM have different time step lengths, necessitating time interpolation between two consecutive coupling fields). Also, requirements for offline coupling should be met, i.e. provision of coupling-input from archived data.

*"The PRISM System should be able to control dynamic model execution, i.e. one or more component models may be launched and/or finished at pre-determined points in the simulation." - Essential*

This is needed for "time-slice" type experiments in terms of a regional simulation within a global model. The same could apply for e.g. AGCMs. Chaining different static simulations can probably fulfil these requirements. In any case, this option should be open for post-PRISM.

There are also potential applications for complex coupling. For instance by running alternatively (with data exchange between them) a coupled ocean/atmosphere plus a simplified chemical model and a coupled atmosphere/chemical model.

*"The PRISM System should be able to control conditional model execution, i.e. one or more component models may be launched during the simulation only if a particular scientific condition is met." - Desirable*

This might be useful in the future. However, in the atmospheric chemistry component, it could also be handled internally. As in the previous paragraph, there are potential applications for complex coupling.

*"The PRISM System should be able to control a global coupled system flexible in terms of executables (extremes are: each component is a separate executable, or: all components run in parallel or in sequence within only one executable)." - Essential*

Some pairs of components will often run in the same executable (ocean + sea-ice for instance). However, the interface definition/implementation should be independent of this aspect. Important, since there is a large range in the computational costs of atmospheric chemistry components.

This offers potential for adaptation to different coupled model configurations using components of different complexities. Note: Presently, in the SMHI coupled RCM all models run in a separate executable, using OASIS.

*"The PRISM System should be able to give some statistic on the load balancing of the run." - Desirable*

For run-time performance.

### **Coupling Exchanges Management:**

*"End-point data exchange: when producing coupling data, the source model should not need to know what other model will consume it; when asking for coupling data a target model should not need to know what other model produces it." - Essential*

This allows for flexibility and easy check of coupling consistency. Although not absolutely necessary, information about sources and targets will allow adaptation to known biases and errors in some models.

### **Termination and Restart:**

*"The PRISM System ensures that the whole simulation shuts down cleanly (both for regular and unforeseen termination) in an intelligent way (e.g. after a restart file is saved), and an error report is generated if one component aborts." - Essential*

A condition for facilitated debugging or continuation of an aborted job.

*"After a machine breakdown, the PRISM System ensures automatically a proper restart of the coupled system." - Essential*

Should be pursued if technically feasible, but not necessarily automatically. Same as in previous paragraph, but continuity in the flow of outputs from the components should also be ensured. The atmospheric chemistry models require correct initialization upon re-start, otherwise the results will not be reproducible.

#### **Other controls:**

*"The PRISM System warns the user if the coupling and I/O frequencies are not synchronized." - Desirable*

#### **Transformer Requirements**

*"The PRISM coupler should provide the following transformations (OASIS transformations are given as examples):*

#### **Time Operations:**

##### *Time Averaging - Essential*

To avoid these calculations in the atmosphere component, but this also implies to transfer more data to the coupler. This could be useful for example over a finite coupling (GCM to RCM) time interval.

Accumulation (e.g. of concentration changes, convective mass fluxes, turbulent fluxes, precipitation, radiative quantities) over time is required, since budgets are very important in chemistry.

##### *Time interpolation - Essential*

It is important to avoid these calculations in the components (for instance wind interpolation for transport of species in the chemical component).

Another example would be the interpolation between two time steps or archived time levels of a global simulation to the time level of the regional model for regional simulation boundary conditions.

##### *Minimum or maximum over a certain time range - Maybe*

This appears to be only diagnostics that could be calculated within the component models. However, one could think of uses for this, e.g. in vegetation models where exceeding a critical temperature leads to a collapse of a population. In ocean biogeochemistry models, exceeding a critical salinity might have similar effects. If information from a global model arrives less often than every time step, absolute minimum/maximum values might be missed unless this feature is present.

Another application is to check maximum Courant numbers for advection schemes that are Eulerian rather than (semi-)Lagrangian.

##### *Other time operations - Maybe*

Spline interpolation might be introduced.

The land surface-modelling group also needs an operator to perform a time sum and time variance calculations. The last one could also be obtained by the operations listed, but it would be simpler if it can be accessed directly.

Calculation of the median may be very useful when we use highly variable non-Gaussian distributed meteorological quantities such as diffusion coefficients or cloud parameters.

## **2D Spatial Interpolation:**

*Nearest-neighbour (Ex: NNEIBOR) - Desirable*

*Nearest-neighbour Gaussian weighted (Ex: GAUSSIAN) - Essential*

Note: This is presently applied in the SMHI RCM using OASIS.

*Bilinear (Ex: BILINEAR) - Essential*

*Bicubic (Ex: BICUBIC) - Essential*

*First order conservative remapping (Ex: SURFMESH) - Essential*

Essential for atmospheric chemistry (converting 2-D emission distributions)

*Second order conservative remapping - Essential*

*Higher order conservative remapping - Essential*

*Remapping using user-defined remapping info (e.g. runoff remapping) (Ex: MOZAIC) - Essential*

Essential for weighted interpolation.

*Other - Maybe*

## **3D Spatial Interpolation:**

Note: At least one 3D interpolation is essential to the GCM-RCM coupling.

*Nearest-neighbour - Essential*

*Nearest-neighbour gaussian weighted - Essential*

*Bilinear - Essential*

*Bicubic - Essential*

*First order conservative remapping - Essential*

*Second order conservative remapping - Essential*

*Higher order conservative remapping - Essential*

*Remapping using user-defined remapping info Essential*

Essential for weighted interpolation.

*Other - Essential*

An interpolation using grid cell air mass as weight (linear in pressure) is essential.

## 1D Spatial Interpolation:

*Nearest-neighbour - Essential*

*Nearest-neighbour gaussian weighted - Desirable*

*Bilinear - Essential*

*Bicubic - Essential*

*First order conservative remapping - Essential*

*Second order conservative remapping - Essential*

*Higher order conservative remapping - Essential*

*Remapping using user-defined remapping info - Essential*

Essential for weighted interpolation.

*Other - Maybe*

## Other transformations:

These should only be local transformations, and any algebraic transformation should be allowed, e.g. *sqrt*, *power 2*, ...

A masking relative to a variable, i.e.  $\max(\text{var}, 0)$ , keeping only positive values, is required.

Transformations which combine two variables need to be considered, e.g. if moisture transport shall be computed.

For these operations, a higher-level language should be used so that one can deal with fields as objects and not only vectors of variables. As an example, CDAT uses PYTHON.

*Conservation: ensure global energy conservation between source and target grid (Ex: CONSERV) - Essential*

Essential to keep a scientific environment.

*Combination of different parts of different coupling fields or of other predefined external data (Ex: FILLING) - Essential*

This may facilitate sensitivity experiments or regional coupling of atmosphere and ocean.

*Algebraic operations with possibly different coupling fields or predefined external data and numbers as operands (Ex: BLASOLD, BLASNEW, SUBGRID, CORRECT) - Essential*

See previous paragraph for remarks.

*Specific algebraic transformations - Essential*

Here the PRISM community has not agreed on one statement. One argument is that we should find agreement on common interfaces so that such transformations are meaningful, the other argument is that there should be no such transformations at all, since there should be no physical knowledge in the coupler.

*Indexing operations: - Essential*

This may facilitate regional coupling, sensitivity experiments etc.

Useful for regional coupling and for forcing a basin based hydrological model with a grid-based atmosphere.

For atmospheric chemistry probably useless, scattering and gathering (see below) may be useful.

*Spatial "collapse" operations: collapse of any dimension or combination of dimensions by various - possibly weighted - statistical operations (mean, max, min, etc.) - Essential*

Useful for sampling forcing data from a global domain for the finite boundary relaxation zones in a regional model.

*Subspace: extraction of subspaces or hyper-slabs in any combination of spatiotemporal or other dimension - Essential*

Essential for sampling forcing data from a global domain for the finite boundary relaxation zones in a regional model.

Desirable in spatial dimension for chemistry optimization.

*Merge: replace  $n$  dimensions by an index dimension by sampling at  $n$ -dimensional points, e.g. replace a lat-lon-height field with values along a trajectory - Desirable*

*Others:*

Calculation of weight functions (runoff remapping, grid interception etc.).

The following transformations occur frequently in atmospheric chemistry models:

Tracer cell mass to mass/volume mixing ratio or to concentration.

Making 3-D advective air mass fluxes mass conserving.

Inversion of a  $n$ level  $\times$   $n$ level exchange matrix for convective transport.

In principle the land-surface community has no specific requirements on the experimental design or the ability of the coupler. A major point is efficiency since data is exchanged between land surface and atmosphere at a very high frequency. The exchanged fields are only two-dimensional, but there is a huge number of such fields.

All operators should deal properly with undefined variables. This is not only essential for observed data, but also has applications in the modelling world. If e.g. cloud water should be averaged, this should be done only when clouds are present.

It is necessary to maintain the positive definite quality of some variables after interpolation (e.g. for concentrations of chemical species, since most chemical modules will crash otherwise).

**Other coupler properties**

*"The PRISM coupler should be able to give some statistics on the coupling fields (mean, max, min, etc.) Ex: CHECKIN, CHECKOUT in OASIS - Essential*

Extremely useful for debugging.



*"The PRISM coupler should support source and target coupling domains that totally or partially overlap (e.g. global atmosphere with a regional ocean, regional model nested into global model, etc.)" - Essential*

Essential for regional coupling (sensitivity experiments, regional climate simulations etc.).

On the other hand, it is useless for global atmospheric chemistry: it requires global overlap. Coupling to a regional atmospheric model makes no sense in view of the constituent fluxes through the boundaries of the region.

*"The PRISM coupler should automatically perform some basic transformations (units - e.g. Celsius to Kelvin, order of dimensions - e.g. source (x,y,z) to target (z,y,x), etc.)" - Essential*

The amount of these operations should be limited if common interfaces are defined.

For coupler related transformations (grids, ...) but NOT for physical transformations, since there should be no physical knowledge in the coupler.

Desirable options: change direction of axes, change origin of x, y-axis.

*"The PRISM coupler should recognize the type of coupling data (flux, vector, scalar) based on meta-data description, and automatically provide relevant transformation operations (this choice of transformation will be validated or invalidated by the user)." - Essential*

Might be useful to provide 'default' interpolation methods (e.g. conservative for fluxes, non-overshooting for positive definite variables, interpolations appropriate for vector fields when grid-poles occurs etc.).

If applicable to rotation of wind components, e.g. when interfacing a rotated regional model domain with a non-rotated global model domain.

It should also point out inconsistencies (e.g. different units).



## I.4 End User Interface

*This part describes the operational and functional requirements of the user interface (UI).*

---

### I.4.1 Functional Requirements

#### I.4.1.1 Setup / Configuration of an Experiment

This function incorporates the preparation and validation of an experiment resulting in a self contained specification, which can be used for

- Submission
- Comparison with other experiments
- Database storage and exchange
- Experiment identification

Several groups of parameters have to be defined to achieve this:

- Scientific parameters
- PRISM specific parameters common to all experiments
- Execution host specific parameters
- Administrative parameters

Feature	Description	Importance
Experiment identifier	Auto system generated	E
Choice of components	Reject incompatible choices	E
Experiment-specific access control	Allow different users to control the experiment execution	O
Timed execution	Ability to set a time for execution	O
Archiving options	Relevant user information needed by the archiving subsystem	E
Select execution host	Select execution host, relevant run-time parameters	E
Validation of configuration	No submission of invalid setups	E
Specify computing resources	Job limits like cpu-time, number of cpu's, memory, etc.	E
Diagnostic choices	Selection of pre-defined verification/diagnostic/post-processing packages etc.	E
Coupling options	Selection and control of available coupling components, coupling frequency, transformations etc.	E
Input/Output options	For instance post - processing frequency, format, restart files	E
Statistics/ Logging/ Debugging/ Trace options	Parameters related to above	E

Table 3: Features of the GUI

Note that execution host specific parameters are transient between different hosts.

A list of all parameters from all components and their constraints has to be provided. This list should be produced in a validating format such as XML.

Enable incorporation of existing tools to auto-generate setups and parts thereof.

Table 3 lists the features the work package has contributed. Each feature has an indicator for its importance as optional or essential. Essential equals to a deliverable.

### *Experiment identifier*

The ability to uniquely define an experiment and supply a user defined description. It needs to be investigated if a PRISM wide identification or a naming local to the executing PRISM host or a combination is best suited. The identifier, which is likely to be used for identification in the preparation, monitoring, archiving and retrieval process should be in any case system generated.

#### **Design Constraints**

This feature is likely to be different on all sites if not the same archiving system is used (which is unlikely).

### *Choice of components*

The ability to couple one or more components in one or more experiments/ensembles.

#### **Design Constraints for Choice of components**

This feature is likely to be difficult to implement if not all users can access all components or if not all components are implemented on all sites. The choice relates to whether the configuration server is made aware of all PRISM sites installed components or if all sites are expected to implement all components.

### *Experiment specific access control*

The ability to specify individual access control for different functions such as monitoring, execution, archiving etc.

#### **Design Constraints**

A PRISM system wide implementation of a fine-grained security policy is likely to be difficult to apply and manage. In particular, if different interfaces are used to run the experiment, access the data, visualisation etc.

### *Archiving options*

The ability to define specific archiving options.

## *Select execution host*

The ability to select a host for the experiment dynamically. A basic translation between platform specific settings may be provided. It should be noted, that many host specific parameters are transient and on exchange of experiments between hosts are likely to be exchanged with a default set for the chosen host.

### **Design Constraints**

This feature is likely to be useful but it should be realized that individual translations of specific system parameters between different hosts are not maintainable. A local knowledge database should instead provide appropriate defaults for the offered configurations at this site. This requires support from the local site.

## *Timed execution*

The ability to define a time for execution.

### **Design Constraints**

This feature depends on the load/management of the executing host. SMS provides many features to allow timed execution.

## *Validation of configuration*

The ability to prevent mis-configured experiments. This should be transparent to the user.

A check system of this kind can be implemented with a rule based knowledge database. This will provide one of the main tools for later maintainability and flexibility of the UI, which will be very important.

### **Design Constraints**

This feature is already in the PreplFS system.

## *Specify computing resources*

The ability to specify cpu-time, number of processors, memory, queues etc.

### **Design Constraints**

See comment on execution host.

## 1.4.1.2 Specific Requirements - Submission and Control of Execution of an Experiment

### *Functionality*

The User Interface should provide the functionality to remotely submit and control the execution of an experiment. Table 4 lists the features the Work Package has contributed. Each feature has an indicator for its importance as optional or essential. Essential are deliverables.

<b>Function</b>	<b>Description</b>	<b>Importance</b>
Submit/start options	Queue experiment for execution, start options	E
Restart	Restart from database/archive/restart files, restart from temporarily suspended state	E
Pause/Interrupt	Interrupt run of experiment (perhaps dependent on specific condition - see "stop") The difference to "stop" is that it is intended to continue the run within a certain time frame or without changing parameters. This could imply for example to keep certain files online and let them not migrate onto tape.	E
Stop (+clean)	Let experiment run until a specific condition is met; manual or automatic shutdown. Especially the scientific and technical conditions may become important as it is intended to let inexperienced users run experiments. Stop at restart points. Stop at specific scientific condition (e.g. temperature difference between ocean and atmosphere at specific exceeds certain limit) Stop at specific technical condition – e.g. disk is filled up to a certain percentage	E
Kill	Stop experiment immediately	E
Modify	Modify parameters, scripts, etc. at run-time.	O
Manage	Manage experiments within a team. I.e. allow access to features to be shared amongst users.	O

Table 4: Job Control: Functions and the Graphical User Interface

### *Design Constraints*

All features are available with the SMS/XCDP tool by ECMWF. However, all of these features are dependent on the capabilities of the model implementation. If these features are to be implemented, all model developers have to program to make the models re-startable etc. A further complication arises for coupled models where the coupling mechanism may have specific requirements. Expertise needed from other work packages. The management and modifications of experiments at run time has severe security implications and may not initially be implemented for that reason in a distributed environment.

## 1.4.1.3 Specific Requirements - Monitoring of an experiment

The UI should provide the functionality to remotely monitor the progress and the status of a running experiment. Ideally, the UI should provide a view of the structure of the experiment.

Feature	Description
Notification of events	User notified of status changes: aborts, completion etc as they occur
View experiment	View of experiment progress and structure

Table 5: Feature and Capability Listing

The UI should notify the user in case of a specific event. Notification might be a message in the user interface, email, SMS, pager ...

### *Design Constraints*

A web interface is highly desirable but security restrictions may limit the technically possible functionality. If possible a graphical view should be provided.

#### 1.4.1.4 Provision of Results of an Experiment

##### *Functionality*

The UI should provide the functionality to access the results of an experiment. Results are defined below.

Result	Description
Scientific diagnostic output	Output from model, see wp2c
Statistics	System use etc. should be defined to be compatible/comparable between hosts
Raw data	Data output from model (components). Realistic sizes must be defined
Logfiles	Output listings for each task of an experiment
Other	%

Table 6: Results and the GUI.

#### 1.4.1.5 Visualisation of Experiment Results

##### *Functionality*

The UI should provide the basic functionality to visualise the results of an experiment. Requirements with respect to the GUI are listed in Table 7 below.

##### **Design Constraints**

Ideally the visualisation and archiving functions should be integrated with the configuration user interface and have the same look and feel.

Issue	Description
Integration	This defines the look and feel of using "one" tool as well as dictates the command and data interfaces
Realisation technology (applet, html, platform-specific)	This is related to the integration issue and the interoperability
Archive access	If visualisation of archived material is needed then some archive browser needs to be provided.

Table 7: Access and the GUI

#### 1.4.1.6 Access Control and Security

##### *Functionality*

The UI has to provide a sufficient security level, which has to be defined. This is linked to the system security model and the features this model should provide.

This project involves many partners with different security requirements. There is a potential conflict between usability and high security. Realistic levels of both are essential. The following terms are used in the text:

Term	Definition
Authentication	Establishing the unique identity of the user
Access control	Defines who can do what.
Security	Preserving the authentication and enforcing the access control.
Accounting	Monitoring of system usage of users/groups

Table 8: Security – terms and the GUI

It is important to understand that the implementation of a system security model is not a function of the UI. As an example, the UI applies the security model by encrypting communications on specific operations. For every operation the necessary levels of security should be defined. The administration involved in the chosen levels should be evaluated.

Feature	Description
Authentication	Implement the security model
Message security	The status of message content. Encryption or validation? Can the user choose the desired level?
Access control	Provision of an access control interface
Definition of user groups	See above
Provide events	And who can use them???
System accounting	???
Override access control	Setting access control on individual experiments???

Table 9: Authentication and the GUI



## *Functional Requirements*

### *Authentication*

The ability to uniquely establish the identity of the user. An Interface to provide a unique token is needed. By message box or file browser for example.

### *Message Security*

The ability to define the level of security in the communication, i.e. switch on encryption. Do the communications in general need encryption?

### *Access Control*

The ability to define who can do what, where and when. Is there a central administrator? Do we need to develop an access control interface for the USER INTERFACE? Fine-grained control should be an optional feature.

### *Definition of User Groups*

Related to above

### *System Accounting*

The ability to monitor the system usage of users/groups.

### *Override Access control*

See access control.

### *Operations and Security Level Required*

All operations require authentication. Further levels such as encryption could be introduced.

<b>Operation</b>	<b>Security required</b>
Connecting to site	Authentication
Loading experiment definitions (own and others)	Authentication
Submitting experiments	Authentication
Monitoring experiments	Authentication
Controlling experiment runs	Authentication
Access results	Authentication
Accessing archived experiments	Authentication

Table 10: Security level of operations

#### 1.4.1.7 On-line User Documentation and Help System

##### *Functionality*

The USER INTERFACE should provide an online documentation and help system.

An extensive help system is required for the functions of the user interface and for the individual components of the coupled system and their respective parameters. This may include the presentation of source code of individual components and/or a scientific description thereof to authorized users.

The help system should provide help on different levels of experience ranging from a complete novice up to an experienced user.

#### 1.4.1.8 Support for model component development

##### *Functionality*

It is desirable that the UI provides support for development of individual model components or the coupler. This may include source code management, compilation and linking, merging of modifications to default libraries, etc..

### 1.4.2 Operational Requirements

Operational requirements with respect to maintenance of access control, users and versions, site specific defaults and implementation of changes to any specifications in experiments, definitions, protocols etc. needs to be defined. A key element of the PRISM system is to allow for system management of models and scripts remotely, i.e. while not being logged in to the site where the model runs. This requirement can be met in the same way as modelers configure models, through the same user interface where the administrators can change system- behavior and versions. The administrator specifies the sources for the model code and its scripts and submits a build job to the PRISM sites, which will then run a generic model build suite using these sources.

## II - Design Options for the PRISM Architecture

### II.1 PRISM System General Architecture

*The purpose of the PRISM system is to enable users to perform numerical experiments, coupling interchangeable model components, e.g. atmosphere, ocean, biosphere, chemistry etc., using standardized interfaces as outlined in REDOC 1.2. The general architecture provides the infrastructure to configure, submit, monitor and subsequently post process, archive and diagnose the results of these coupled model experiments. There is an emphasis on choosing an architectural design that allows these activities to be done remotely, e.g.. without the user physically being in the place where the numerical computations take place. The required features of the PRISM system are analysed with respect to the processes involved, the actions they take and where they happen. The general architecture influences the security models that can be applied. It defines the possible operations of a user from a remote site. An architectural design is presented that satisfies the security and configurability demands required by all processes. Existing technologies are investigated to assess the constraining implications of their use. It is expected that the cost and capacity of future computing and network technologies are changing. Therefore, it is important to design an adaptable and scalable architecture.*

---

#### II.1.1 Terminology and Concepts

##### *Experiment*

An experiment is an ensemble of tasks running on a supercomputer, defined by a configuration process. Three levels of communication exist within such a coupled experiment:

- Macro-dependencies and triggers between different tasks (e.g.. handled by a scheduler like SMS)
- Dependencies and triggers between different component models and the coupler (e.g.. handled by OASIS, PALM)
- Inter process communication within each model component (e.g.. MPI, OpenMP, etc.)

The first level of communication is under the control of a scheduler. The function of the scheduler is to co-ordinate the execution of these tasks while preserving any dependencies between them.

##### *Task*

A task is an individual job step of an experiment that needs to be executed. The Figure 1 is a hierarchical view of an experiment with the tool Xcdp showing a collection of tasks. The different boxes represent different tasks and the colour code shows the status of the task. Note, that the coupled model, i.e. coupler and component models, represents a single task of an experiment.

## Coupler

One major objective of the PRISM project is to develop a standard interface for the models constituting the global climate system. The models will exchange information with a universal coupler or directly with the other model components. The coupler is the program responsible for controlling the coupled model formed by the different component models, and controlling the exchanges and transformations of physical data between them. This is detailed further in REDOC II.2.

## Configuration

Three basic phases of configuration can be identified:

- **Definition** of all entities of a coupled experiment
- **Composition** into a specific coupled experiment
- **Deployment** of the coupled experiment onto a set of computing resources

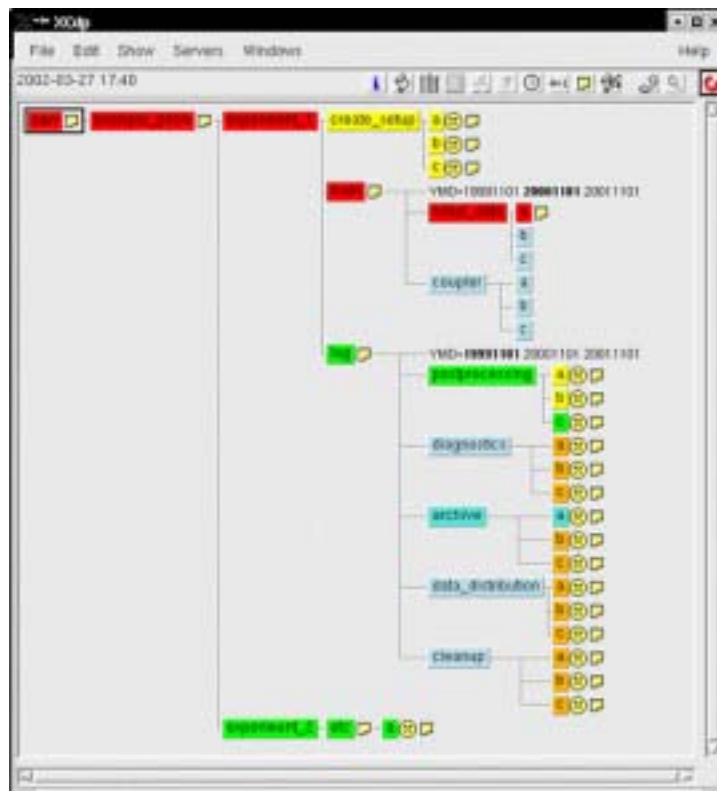


Figure 1: Xcdp graphical view of an experiment, the status of tasks indicated by different colours gives an overview of an experiment.

The **definition** phase comprises the definition of all component models to be coupled (model interfaces and meta-data - PMIOD), transformation entities, I/O options, post processing options, diagnostic options, statistic options, ... etc. In the PRISM system this is provided by the PRISM model administrator and presented to the PRISM user through the user interface.

During the **composition** phase the PRISM user sets up a specific coupled experiment through the user interface by

- Selection of individual model,
- Configuring the constitution of each individual model component (SMIOC),
- Composition of the coupling configuration (SCC),
- Selection of other pre/post processing options,
- Selection of the site and computing resources to use.

During the **deployment** phase an abstract compact description of an experiment is generated. This is defined as a configuration instance. A configuration instance details how to run the coupled experiment on a computer in a format that can be understood by the computer's operating system. Further, it contains information on the coupling communication between models and the internal communication of each model component on the chosen platform. This is further detailed in REDOC II.2. Consistency checking before deployment ensures a correct configuration for each task.

### *User interface*

The system should allow the domain activities by remote access, i.e. the users do not have to be physically in the same place as where the model is executed or data is located. The interaction between the users and the system takes place through a user interface. This interface establishes the identity of the user and allows for access to the systems functionality. The functionality is provided by a number of specialized servers accessed by the client user interface (UI) detailed in REDOC II.4.

### *Administration*

Each institution will name an administrator, maintaining and developing a particular model component of the coupled PRISM system. The administrator has several tasks:

- Build of new versions of individual model components on all or selected PRISM sites.
- Provision of build mechanisms for model components.
- Provision of the definition of all entities available for a coupled experiment.
- Provision and maintenance of a local repository.
- General maintenance and support.

This is accomplished through an administration user interface. Since there is a need for varying expertise to accomplish all administration tasks, the administration may be performed by several persons. Due to the likely distribution of expertise for the different model components, administrators will be physically located at different sites, which further emphasizes the need for a distributed system.

### *Results*

The experiment results in the output of data fields, statistics and diagnostics. This output needs to be archived, catalogued and made accessible to the modeller, who needs to visualize the diagnos-

tics and data to understand the results. It is the task of the archiving and data management system to accomplish this and the details are explained in the REDOC II.3 documents.

## *Client and Server Processes*

The partitioning of functionality allowing a client to perform operations outside its own capability with the help of a more powerful server is called client/server computing. The client and the server may reside on physically different computers and they communicate by accessing computer networks.

### II.1.2 System model

#### *System actors and their activities*

Three actors on the PRISM system can be identified: Users, developers and administrators.

It is important to distinguish between the behaviour of these actors:

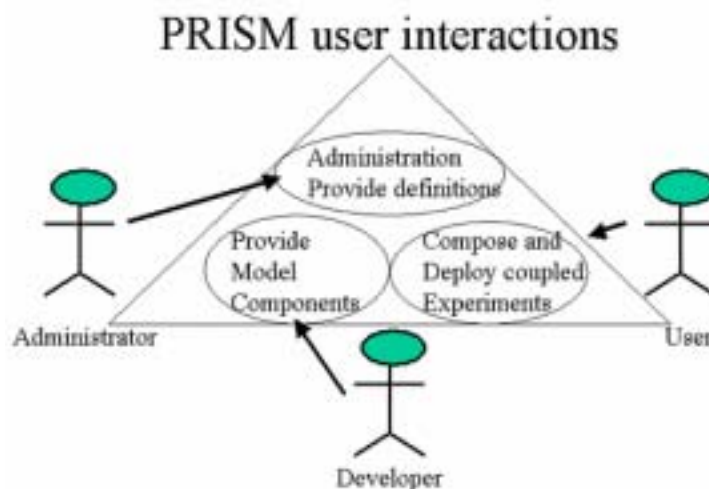


Figure 2: PRISM user interactions

- **Users:** Will run experiments by adding or replacing parts of a given model code or configuration. Their requirements are ease of use, to be shielded from implementation details, traceability of experiments, collaboration with other modellers in a standard way, help systems available, template setups, error checking, ease of deployment.
- **Model developers:** Will develop and test new models and configurations. Their requirements are full control of the system including sources, versioning of sources, minimal overhead when testing new features, no restrictions or dependencies on other users.

- **Administrators:** Will provide the link between developers and users. Their requirements are to start administration tasks on local or remote sites infrequently, ease of use, and automatic model management.

PRISM-Actor	Main activity and interaction with system	Acts on
Administrator	Executes administration tasks	Definitions of administration entities
Administrator	Provides definitions of all entities	Model component interfaces and meta-data
User	Composes coupled experiments	Selection of all entities
User	Visualizes, queries and manages	Model results
Developer	Develops model components	Model components

Table 11: PRISM actors and main activities

The groups represent diverse and contradicting demands on the system and one system cannot satisfy all of the demands. We therefore suggest that the system will be developed in two phases and into two different products, which are configured from the same software base:

- **Developer system:** Will consist of mainly the User Interface decoupled from the web services infrastructure and its resources. Represented by Architecture C, Figure 5 on page 50.
- **User/Administrator system:** Will consist of the UI connected to the web services infrastructure and enabling support from its services such as templates, documentation, remote submission etc. Represented by Architecture A, Figure 3 on page 49.
- **Users system:** Will work in isolation and allow the developers to carry out their work locally with full control over the resources.

The exact differences will be outlined in the ARCDI part of the specification.

## *Process View*

The actors realize their activities by means of a user interface and the following processes can be identified from the activities above:

1. Client configuration processes
2. Configuration provider processes
3. Execution processes

A more detailed list of activities from above, grouped by the processes 1, 2 and 3, is given in

Table 12 on page 48. Each process is either distributed over the participating PRISM sites or local, noting that initially different model components are not to be distributed in the system.

<b>Table of client configuration processes (1)</b>	
Location of activity	Activity type
User interface	Configuration
	Visualization
	Authentication
	Archive query
	Documentation
	Configuration instance
	Monitoring
<b>Table of configuration provider processes (2)</b>	
Configuration server	Configuration
Configuration server	Configuration instances
Documentation server	Documentation
Authentication server	Authentication
Administration server	Model build configuration
Administration server	Model build configuration instance
Experiment database server	Configuration instances for the experiments
Visualization server	Visualization
Monitoring server	Start/stop/state information
<b>Table of execution processes (3)</b>	
Scheduling server	Configuration instances
Execution server	Coupled model (coupler + component models)
Execution server	Data pre/post processing
Archiving server	Archiving

Table 12: Table of different processes

### *Proposed architecture*

The client configuration process is accessed through the Internet. The configuration provider processes are accessed through a central site but the services can be distributed to other sites without functional difference. The execution process is local to the model provider. This is described as directory centric, web enabled and distributed from local PRISM sites.

Variations of the architecture are shown below on page 49 and 50 in three figures. They show a Central PRISM site and a Local PRISM site. The local site is where the execution, scheduling and archiving server is located and the central site is one of the participating PRISM sites where the configuration provider processes listed in

Table 12(3) are located and used by all client processes.

The component boxes in the figures represent the following:

Administration Interface - Access point to Administration Server

- Administration Server - Serves build configurations to Administration interface.
- Central Repository - Collection of data or programs such as configurations, land and sea mask etc. used by all clients.
- User Server - Contains configuration provider processes accessed by the client.



- Model execution - Processing of the coupled experiment at the PRISM local site.
- Local repository - Collection of common data or programs such as configurations, land and sea mask specific for the local site and the deployment environment for experiments set up by users to run on the site.
- Controlling instance - Scheduling of an experiment.

The figures show the architecture when components are moved from the central PRISM site to the local Data view.

The data in the system consists of:

- Configurations
- Configuration Instances
- Model input
- Model results.

The architecture has to provide for the movement of this data. A data access policy and a security policy determines who can move what data and where it can be moved. This requires interaction with many subsystems such as archiving processes and network providers.

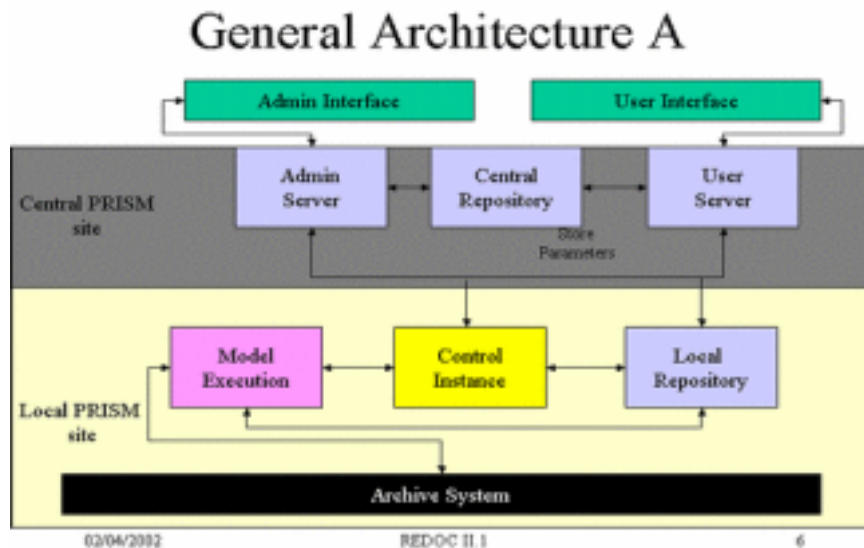


Figure 3: Central Site Architecture, directory centric

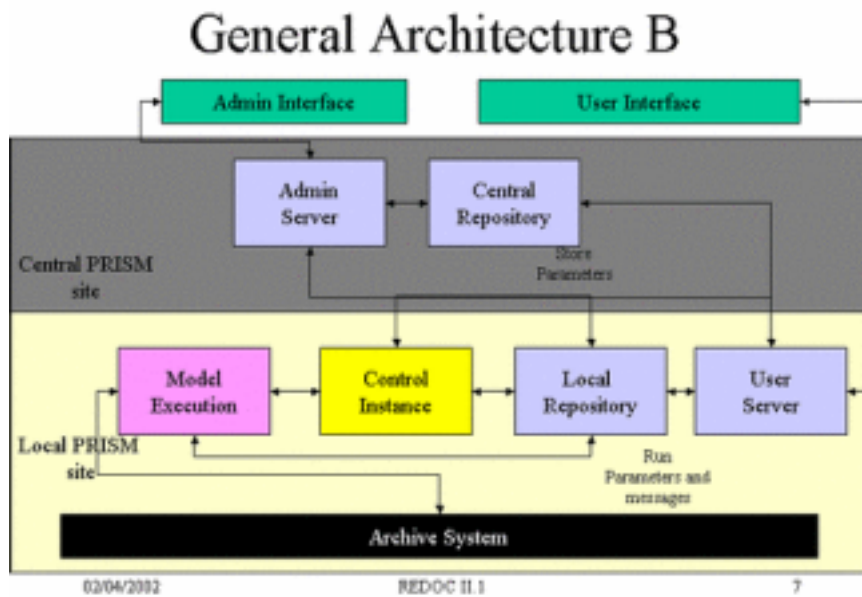


Figure 4: Common Data Architecture, model provider centric

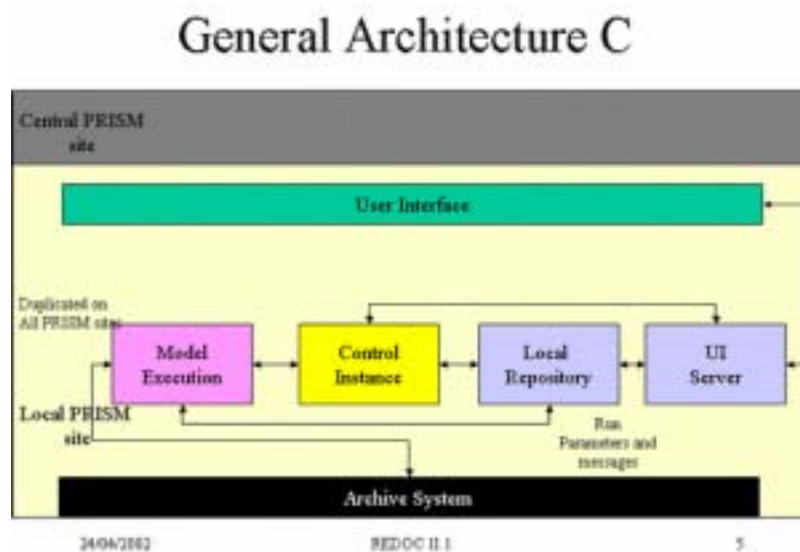


Figure 5: Full replication architecture, no central repository

### II.1.3 Software life cycle

Inter operating components in collaborative and distributed environments are deployed and maintained by multiple administrators and thus upgrades and maintenance is likely to be uncoordinated. With no central authority to plan and execute upgrades some clients will always be out of synchronization. This applies to the scientific models as well as the computing model, i.e. the

infrastructure components such as application servers. For the computing model a practice of allowing the different software (component model) providers all to act as administrators enabling them to start distributions of upgrades to all sites should solve this problem.

For the computing model infrastructure software such as service providers, the problem is complicated by the fact that services can call each other. A service should:

- Detect incompatibility with calls
- Find service providers
- Authorize and authenticate with provider
- Transport and install new versions
- Register and announce new versions
- Release old versions and clean up.

Figure 6 represents a possible architecture where services are deploying themselves through a central service provider. Authentication and security mechanisms must allow the service provider to provide the new versions of software for automatic deployment. Failure of supplying the semiautomatic life cycle management will lead to service failures and increased administration costs.

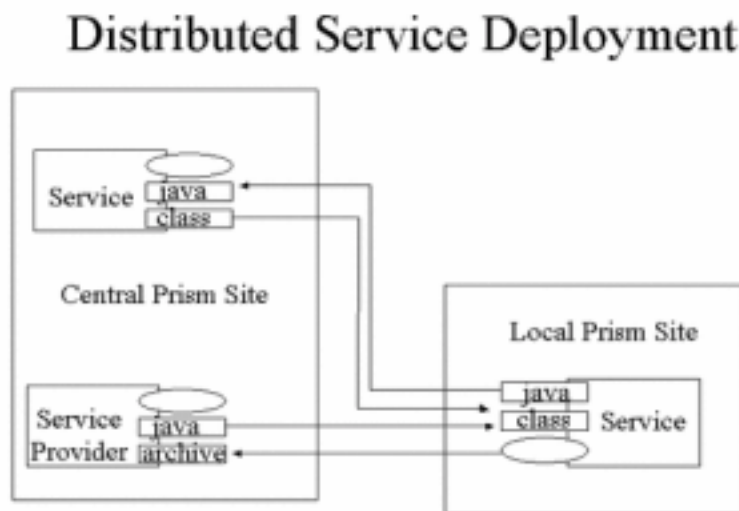


Figure 6: PRISM software deployment model

## II.1.4 Security

### Overview

System security is made up of the following components:

- Authentication - Proof of identity
- Authorization- The user can only see information he is allowed to

- Confidentiality - Information transmitted is not read by third parties
- Non-disputable - The sender cannot deny that the message was sent
- Integrity - Data transmitted is not tampered with

Authentication can be made by:

- Intellectual property, something you know such as a password
- Physical property, something you have such as a certificate
- Biological property, something unique to you such as a fingerprint

The strength of security is normally determined by the factors involved: i.e. password is one factor, certificate is a second. Combining the two raises the strength by magnitudes.

Authentication normally means that two computers can verify the identity of each other, not that the operators are who they claim to be. The combination of biometrics (fingerprints for example) together with physical property is a very strong authentication and could solve this problem. A different form of authentication is when you need to authenticate between two computers without any operator intervention. This situation arises frequently in a services oriented system when two services are dependent on each other or when new operations are instigated by a service requiring further authentication.

There are in principle the following security solutions in operation today:

Password based - These solutions require an operator to supply the password. Various levels of sophistication involving rotation, time of validity and reuse of passwords can be found. One example is s/key system (<http://www.freesoft.org/CIE/RFC/Orig/rfc1760.txt>).

Physical token based - These solutions require you to hold a certificate, smart card or similar. Various commercial and free offerings available building on X509 (<http://www.openssl.org/docs/apps/x509.html>) certificates or proprietary smart cards.

Public key/Private key solutions. These solutions build on the public and private keys being able to encrypt and decrypt messages thus verifying each other. Can be combined with password or physical tokens. Offers encryption of communication. Systems building on these are Secure Shell (SSH, <http://www.openssh.com/>) and Kerberos (<http://web.mit.edu/kerberos/www/>).

## *Requirements*

There should be an access model enforced in the system to ensure proper authorization. The level of control is to be set to be practical in terms of administration and confidentiality and to be determined by the PRISM partners.

The levels of integrity in system transmissions is to be high but the confidentiality is not so important as messages will mainly consist of configuration information which is less useful if you have no access to the software being configured.

The service-to-service authentication needs to be solved in a scalable manner consistent with the administration resources available.

## II.1.5 Specific Design Choices and Constraints

### *Design Choices*

- Highly modularized system where parts can be removed or duplicated as conditions change.
- Allow component exchangeability and portability.
- Service based concept enabling new services to be discovered and old replaced.
- The use of sufficient security and access control level.
- Minimum maintenance solutions.
- Allow for distributed execution if possible.
- Use standardized protocols and existing software.
- Align with other projects.
- Common look and feel for all components.

### *Constraints*

- High security might give poor usability.
- Security may implicate increased maintenance.
- Security limitations on different sites.
- Access control may be difficult due to using distributed components with different capabilities.
- Low maintenance costs and available administration staff.
- Short time frame does not allow much development and coding time and "off the shelf" components should be used bearing in mind that this limits control over standards and look and feel.
- Look and feel of software may be less important if the software is far superior.

## II.1.6 Software Packages Investigated

*PALM* : [http://www.cerfacs.fr/globc/PALM\\_WEB/index.html](http://www.cerfacs.fr/globc/PALM_WEB/index.html)

The PALM project aims to provide a general structure for a modular implementation of a data assimilation system. In this system, a data assimilation algorithm is split up into elementary "units" such as the observation operator, the computation of the correlation matrix of observational errors, the forecast model, etc. PALM ensures the synchronization of the units and drives the communication of the fields exchanged by the units and performs elementary algebra if required. This goal has to be achieved without a significant loss of performances if compared to a standard implementation. It is therefore necessary to design the PALM software in view of the following objectives and constraints:

**Modularity:** PALM provides a mechanism for synchronization of pre-defined functional units that can be executed in sequence, in concurrence, or in a mix of these two modes. One key aspect of PALM is also that dynamic or conditional executions of these units are allowed. PALM also performs the required exchange of information between these units. The user through a sophisticated Graphical User Interface defines the configuration of a Palm run.

**Portability:** PALM aims to run on all the existing high-performance platforms and, if possible, on the next generation supercomputer. This effort of "clairvoyance" can be accomplished only through the adoption of standard.

**Performances:** PALM will be used in two modes: research and operational. The research mode will be used for the design of new algorithms and will prioritize the flexibility; on the contrary, the operational mode will work with a fixed configuration of the algorithm and will prioritize the performances optimization and the monitoring of the process.

Palm should be considered as a coupler designed for dynamic coupled simulation. The Palm GUI (Pre-Palm) is a sophisticated GUI required to describe a dynamic simulation and the complex relations that can occur between the different dynamic units in the task. For static runs this level of sophistication is probably not required and can incur performance penalties.

**UNICORE:** (<http://www.unicore.de>)

UNICORE is meta-computing framework based on an Abstract Job Definition that can be submitted to different sites from Java (<http://www.sun.java>) clients. Gateways receive the jobs and translate and schedule the definition for execution on the available hosts. Security is based on certificates. The client is downloaded once together with definitions. Requires programming skills to develop new job definition interfaces (plug-ins) and takes considerable effort. Unicore mostly lacks comprehensive scheduling and monitoring mechanisms and is not used in a production environment yet.

**GLOBUS:** <http://www.globus.org/>

The Globus Project is a multi-institutional research and development effort creating fundamental technologies for computational grids. Grids are persistent environments that enable software applications to integrate instruments, displays, computational and information resources that are managed by diverse organizations in widespread locations. A primary product of the Globus Project is the open source Globus Toolkit, which is being used in numerous large Grid deployment and application projects in the United States, Europe, and around the world.

Parts of the Globus project software relates to the tasks at hand in PRISM, such as security mechanisms (certificates), resource lookup, scheduling and Message Passing (MP) technology. Benefits are that many important institutions and commercial interests are supporting the Globus initiative.

## *PreIFS/SMS*

PreIFS is an interactive meteorological application to prepare research experiments using the integrated forecasting system (IFS) at ECMWF. Both researchers at ECMWF and scientists in institutions anywhere in Europe (subject to prior permission) can access the complex computer envi-

ronment at ECMWF via the Java application PrepIFS or via the INTERNET using the Java-Applet PrepIFS and any standard WWW-browser. Forecast/Analysis Experiments can be prepared and submitted remotely.

The system uses a combination of web servers and application brokers/directories/providers to communicate with the preparation client application, which contains functionality to validate the prepared experiment before it is submitted for processing.

Supervisor Monitor Scheduler (SMS) is an application that enables users to run a large number of programs which may have dependencies on one another, and in time, in a controlled environment with reasonable tolerance of both hardware and software failures, combined with good restart capabilities. SMS submits tasks and receives acknowledgements from the tasks when they change status and when they send events. SMS knows the relationships between tasks, and is able to submit dependent tasks when a given task changes its status, for example when it finishes. An associate application Xcdp allows you to monitor and change jobs in the scheduler in a GUI. The scheduling application is currently only used within local area networks.

## *SSH*

SSH, Secure Shell is an authenticating protocol used for remote host access and is very secure. It works with public and private key authentication and encrypts transferred data. It has commands for ftp and login and may be a useful tool for administration.

## *Technology trends*

It has been suggested [Foster 2002, page 191] that the rate of technology change, i.e. the rate at which capacity doubles or price halves, are around 9, 12 and 18 months for networks, storage and computing power. If network performance doubles relative to computing power every 18 months it will become essentially free. From this point of view it is important to select an architecture that can exploit this advantage.

### II.1.7 Discussion on the architectural choices

The best designs in order to achieve remote access, modularity and extensibility are the directory centric (A) and the model provider centric (B) architectures as outlined in section Proposed architecture.

The directory centric (A) architecture benefits from that it minimizes the duplication of static or semi static resources (i.e. land and sea mask). It also allows central content to grow but local content can still be chosen if appropriate. For deployment, PRISM sites do not need full web and application servers that makes management easier. Future co-operative techniques can be used from the central site, such as client visualization displaying on many clients.

The drawback of the directory centric (A) architecture is that the complexity increases as resources needs to be advertised and discovered by clients. Some concentration of processing power may be required to serve all clients.

The final architecture will show that combinations of local and central resources are possible, as they will not compromise the system.

The administration effort required for the central site architecture is likely to be less as the duplication of data and software is not necessary and thus fewer physical copies need to be accessed.

It is important to understand that most of the system service communication is made over the Internet, a network over which we have not full control. As a result response times will vary considerably for messages and the actions invoked through the user interface. Recoverability is limited as it is often difficult to diagnose where errors occur. If a certain level of performance is deemed essential a virtual private network should be set up with a service level agreement.

## II.1.8 Cooperating systems

### *Technology*

The technology that realizes the proposed architecture is known as "Web Services". This includes the use of web servers, application servers, resource directories and discovery mechanisms and message services and the use of Java clients and servers. For security mechanisms certificates and Secure Socket Layers (SSL, <http://www.openssl.org/>) as well as encryption can be used. Web services as a technology are service centric, allowing clients dynamic service discovery over networks such as the Internet. It is usually deployed as a three tier system involving a front end presentation layer such as a browser or java client communicating with a remote domain application (service) through a web server. The web services infrastructure will see benefits coming from application integration of diverse software made possible by standardization and directory technologies to enable service providers to publish their services irrespective of implementation technology.

### *Standards*

The issue of standardization of interfaces in complex and configurable systems becomes very important in deploying distributed architectures for scalability, extensibility and future success. A key factor making the inter operability between software possible is the development of XML, the eXtensible Mark-up Language (<http://www.w3.org/XML/1999/XML-in-10-points>). This language allows for standardization of messages between systems enabling clients and servers to inter operate over networks. The development of XML promises to standardize several other important technologies such as:

- Remote Procedure Calls - Making it possible for one program to call other programs running on remote hosts and using different programming languages through protocols like SOAP (Simple Object Access Protocol, <http://www.w3.org/TR/SOAP>).
- Resource descriptions - The Web Services Description Language (WSDL, <http://www.w3.org/TR/wsdl>) makes it possible to describe resources such as documents or procedures with XML and to allow this information to be incorporated in directories.
- Resource lookup - UDDI, the Universal Description, Discovery and Integration protocol (<http://www.uddi.org/>) defines ways to publish and discover information about web services.

In the future this standardization will make it possible for systems to share and exchange information in a structured way. PRISM will be one of the projects ready for the future by the use of these technologies.



## PRISM Infrastructure Software Implementation

It would be possible to implement all server components in any suitable language such as Perl or C++. Currently there is no client software that can be used with browsers that does not build on Java technology. From a system maintenance point of view using one technology, Java, is the preferred way as this simplifies the task of adhering to multiple standards. The best choice is therefore to implement all software in the infrastructure in Java but not to restrict it if there is a case for using other technologies. Java supports all the mechanisms needed for implementing web services using available standards. Other technologies are Microsoft's DotNet and HTML. Today DotNet technology is very new and is also proprietary in nature. The use of a HTML client severely limits the intelligence that can be built into the client and is therefore seen as less useful.

Other projects such as Globus have published similar ideas (Open grid services architecture) building on the web services concept. There is no doubt that the web services concept will be the dominant paradigm over the next 5 years and together with standardization of technologies, increased network speed and co-operative efforts, the systems that are ready for the interaction, will have an advantage.

### II.1.9 Risks

Risk	Risk Magnitude	Description	Impact
Security demands incompatible on some sites.	Severe	Multiple security solutions may be necessary.	If sites cannot agree on one security solution it may introduce costly separate solution affecting the client experience.
Lack of infrastructure resources.	Severe	Hardware and software must be available for web services	Slow or non-existent services.

implementation.

Table 13: Risks



## II.2 PRISM Coupler, Including Coupler Review

*After introducing some general coupling concepts, a list of possible requirements and design options for the PRISM coupler is given. These requirements will clearly not be all fulfilled and these options will clearly not be all implemented in the future PRISM coupler. This exhaustive list of possible requirements and options will help the coupler developers to identify the relevant functionalities for the future PRISM coupler and to establish a list of priorities for the next 3 year developments. A review of existing couplers and coupling applications is presented in the last section.*

---

### II.2.1 Introduction - Definitions

This section introduces some general coupling concepts. The nature of a coupled simulation is first discussed; it can be static, dynamic, or interactive. The possible coupling relations between two component models is then analysed.

#### *Static, dynamic, or interactive coupled simulation*

An important concept relates to the possibility given, or not, to the coupling parameters to evolve during the coupled simulation. The coupling parameters include:

- The component models
- The characteristics of the coupling exchanges (fields, frequencies, transformations, etc.)
- The characteristics of the coupling-fields themselves (units, grid, partitioning, etc.).

Different options can be defined: a coupled simulation can be **static**, **dynamic** or **interactive** (with respect to the process management, or with respect to the coupling exchange characteristics, or with respect to the coupling field characteristics).

#### **Static:**

All coupling parameters are fixed initially and do not change during the whole simulation. All information given by the models (coupling field units, grid, partitioning, etc.) or prescribed by the user (components, coupling fields, coupling frequencies, etc.) is defined only once initially. The component model processes and their corresponding rank and location (processor, node) are fixed from the beginning to the end of the simulation.

#### **Dynamic:**

- With respect to the process management (PM dynamic): Component models and/or additional coupling processes can be launched during the simulation. An example is a coupled simulation that starts with a reduced number of component models, these components first reaching some kind of equilibrium before other components are started. One could also think of a regional coupled simulation starting only with an at-

mosphere and an ocean component models, and in which a sea ice model is activated only if the sea surface reaches freezing conditions.

- With respect to the coupling exchange characteristics (CE dynamic): The characteristics of the coupling exchanges (fields, frequencies, transformations, etc.) are allowed to change during the simulation. One example is a coupled simulation in which a particular coupling field is exchanged only if a particular scientific condition is met.
- With respect to the coupling field characteristics (CF dynamic): The characteristics of the coupling fields themselves (units, grid, partitioning, etc.) are allowed to change during the simulation. Transfer of information between the model and the rest of the coupled system must be possible at any point during the simulation. One example is a coupled simulation in which one or more components are subject to dynamic grid refinement.

### Interactive:

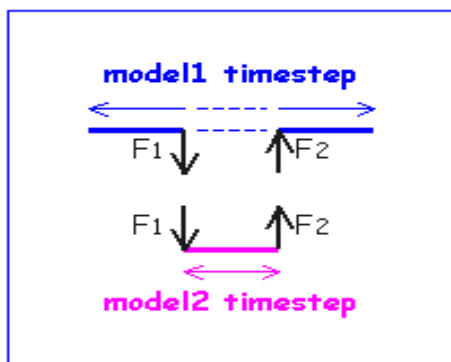
The user can modify the coupling parameters at run-time: an interactive coupling is necessarily dynamic. All implications of a dynamic coupling are also valid for an interactive coupling, with the added implication that the information prescribed by the user must be transferable to the coupled system at run-time.

As a dynamic coupled simulation, a simulation can be interactive

- With respect to the process management (PM interactive)
- With respect to the coupling exchange characteristics (CE interactive)
- With respect to the coupling field characteristics (CF interactive)

### Possible coupling relations between two components

The coupler co-ordinates the execution of several major climate component models. It is firstly important to analyse the coupling relations that can exist between any two of these components.



Two components can be *sequential by nature* or *concurrent by nature*. It is also possible to force two components sequential by nature to run concurrently, or two components concurrent by nature to run sequentially; we will refer to two components having one of these relations respectively as *concurrent by construction*, or *sequential by construction*. The differences are demonstrated on the next pages in Figure 7 to Figure 10.

Figure 7: two models are **sequential by nature** if the first model necessarily waits while the second model is running, and vice versa. The sequence is imposed by the exchange of coupling fields.

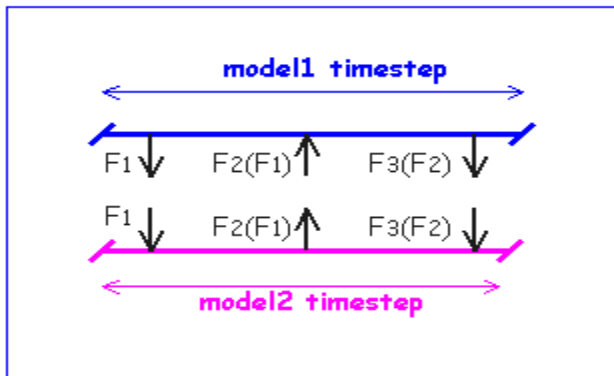
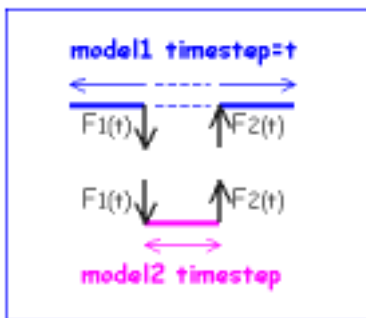


Figure 8: Two models are **concurrent by nature** if coupling data produced by one model depend on other coupling data produced previously by the other model during the same time step, and vice versa

models sequential by nature...



... forced to run concurrently

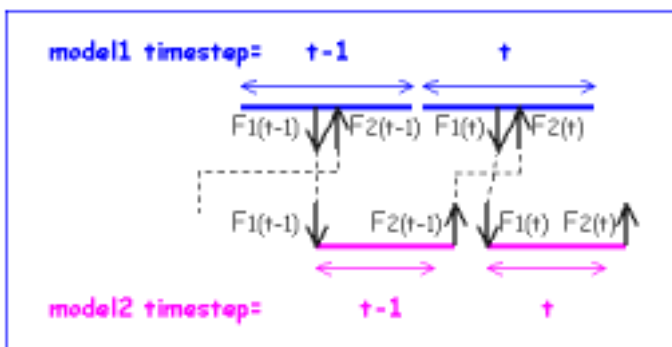
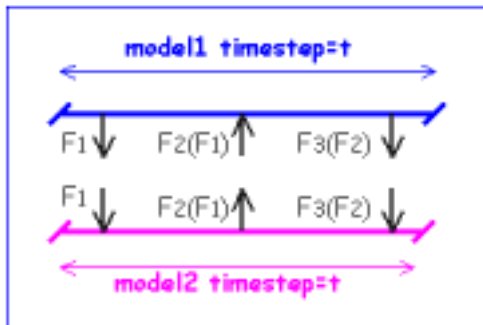


Figure 9: Two models are **concurrent by construction** if they are sequential by nature but forced to run concurrently. This requires, at a given time step, that coupling data produced at the preceding time step are used as input

models concurrent by nature...



... forced to run sequentially

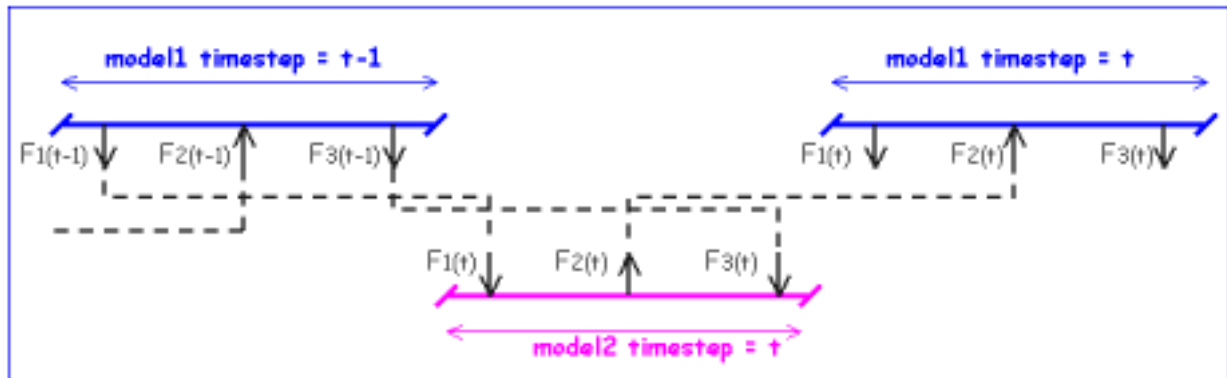


Figure 10: Two models are **sequential by construction** if they are concurrent by nature but forced to run sequentially. This requires, at a given time step, that coupling data produced at the preceding time step are used as input.

## II.2.2 Possible Requirements and Design Options for the PRISM Coupler

The main constituents of the coupler are: the Driver, the Transformer, and the PRISM System Model Interface Library (PSMILe), which interfaces the model with the rest of the coupled system, and therefore includes the Data Exchange Library (DEL).

### *General Requirements*

- The overhead associated to the global system modularity and flexibility is acceptable.
- The whole system is portable and efficient on the different hardware architectures used for climate modeling, on dedicated or shared hardware resources. Standard and portable solutions should be preferred. However, for critical issues for which a portable solution does not exist or would lead to very low efficiency, machine dependent options could be offered.

- The design and implementation lead to code easy to maintain and which can be easily modified to support future model or coupling functionalities.
- Design reflects a clear separation of responsibilities for the different parts of the coupler.
- The PRISM System infrastructure can be used to technically assemble a coupled system based on any component models, even if these models do not conform to the PRISM physical interfaces, given that they include the well-defined PRISM System Model Interface Library.
- The PRISM System infrastructure can be used to couple an arbitrary number of component models; any component can be one-way or two-way coupled with any other component.

### *Driver Requirements and Design Options*

The Driver manages the whole coupled application. It may launch the component models, monitor their execution and termination, orchestrate the exchanges of coupling data, centralize and distribute simulation parameters which require a consistent definition among all component models, and centralize and distribute information on the component model status during the simulation.

A design option is to decentralize the coupling functionalities as much as possible in the Data Exchange library included in the different model interface libraries and in the Transformer, and therefore to reduce as much as possible the role of the Driver. This option is probably applicable only for static coupled simulations and allows an easier evolution toward heterogeneous coupling (different component models running on different machines).

#### **Model Execution and Control:**

- The Driver can control model execution concurrently, in a regular sequence (one after the other), or in some pre-defined combination of these two modes.
- The Driver manages static simulations; this is the minimal option and the simplest one to implement.
- The Driver can control dynamic model execution. As presented above, this functionality will be required for scientific reasons if it is decided that the PRISM System should support dynamic coupled simulations. A discussion on the technical advantages and disadvantages of a dynamic Driver (with respect to the process management) is presented in the Appendix REDOC II.2 .
- The Driver can control conditional model execution (one model is started during the simulation only if a particular condition is met).
- The Driver can control interactive coupled simulations.
- The Driver can control a global coupled system flexible in terms of executables (extremes are: each component is a separate executable, or all components run in parallel or in sequence within only one executable).
- The Driver can take advantage of extra hardware resources as they come available, within a static envelope.
- The Driver can give some statistic on the load balancing of the run.
- The Driver includes a timing that allows it to sample with identical absolute time for all component models the duration of events.

- The Driver warns the user if he tries to assemble an invalid combination of component models.
- The Driver warns the user if the coupling and I/O frequencies are not synchronized.

### Information Management:

- The Driver centralizes the universal parameters, i.e. the parameters that need to be consistently defined in the coupled system (initial date, length of integration, calendar, earth radius, solar constant, restart saving frequency, etc.). This information can be defined by the user or by one master model (the atmosphere). The Driver transfers this information to all component models.  
(In a decentralized approach, the universal parameters would be read in directly by each model PSMILe.)
- The Driver centralizes all model information (grid definition, distribution, etc.). For *CF dynamic* simulation, this information may evolve at run-time. The Driver transfers this information to the rest of the coupled system, when and where required.  
(In a decentralized approach, each model PSMILe would be responsible for transferring the appropriate information to the appropriate processes.)
- The Driver centralizes information on the state of all component models in the simulation and transfers this information to a higher level-controlling layer or to the user.  
(In a decentralized approach, each model PSMILe would be responsible for transferring its status information to a higher level-controlling layer or to the user.)

### Coupling Exchange Management:

- The Driver performs the matching between output coupling data produced from one model and input coupling data requested by another model. In case of static simulations, the matching can be performed initially; for dynamic simulations, the matching will be performed at run-time. At run-time, the Driver manages the exchanges of coupling data based on this matching.

(In a decentralized approach, the matching could be performed initially and the exchange could be managed at run-time by each model Data Exchange library included in its PSMILe. This option is probably applicable only for static simulations.)

The matching and choice of coupling parameters (e.g. coupling frequency) could be performed:

- **Automatically**, when there is only one matching possibility between output and input coupling data;
- **Based on user's choices** indicated in a coupling configuration file.

### Termination and Restart:

- The Driver ensures that the whole simulation shuts down cleanly (regular and unforeseen termination) in an intelligent way (e.g. after restart is saved) and reports error if one component aborts.
- The Driver constantly updates, by writing in a restart log file or by any other equivalent mean, the last date for which all model restarts were saved. In case the coupled system is re-started after an unforeseen termination (i.e. machine breakdown), the Driver automatically finds in the restart log file the appropriate date of restart; it restarts the component models and transfers this restart date to all of them.



- The Driver is able to shutdown the simulation cleanly if a specific scientific condition is met (e.g. average SST exceeds some predefined value).

## *Transformer Requirements and Design Options*

The Transformer performs on the coupling data all transformations required between two component models.

### **List of possible transformations and other requirements**

The transformer may provide the following transformations (words in brackets refer to the OASIS key words):

- Time operations:
  - Averaging or sum
  - Interpolation
  - Minimum or maximum over a certain range
  - Variance
- 2D spatial interpolation:
  - Nearest neighbour (Ex: NNEIBOR)
  - Nearest neighbour Gaussian weighted (Ex: GAUSSIAN)
  - Bilinear (Ex: BILINEAR)
  - Bi-cubic (Ex: BICUBIC)
  - 1st order conservative remapping (Ex: SURFMESH)
  - 2nd order conservative remapping
  - Higher order conservative remapping
  - Remapping using user defined remapping info (e.g. runoff remapping) (Ex: MOZAIC)
- 3D spatial interpolation:
  - Nearest neighbour
  - Nearest neighbour Gaussian weighted
  - Bilinear
  - Bi-cubic
  - 1st order conservative remapping
  - 2nd order conservative remapping
  - Higher order conservative remapping
  - Remapping using user defined remapping info
- 1D spatial interpolation:
  - Nearest neighbour
  - Nearest neighbour Gaussian weighted

- Bilinear
- Bi-cubic
- 1st order conservative remapping
- 2nd order conservative remapping
- Higher order conservative remapping
- Remapping using user defined remapping info
- Other transformations:
  - Conservation: Ensure global energy conservation between source and target grid (Ex: CONSERV)
  - Combination: Of different parts of different coupling fields or of other predefined external data (Ex: FILLING)
  - Algebraic operations: With possibly different coupling fields or predefined external data and numbers as operands (+, -, X, SQRT, ^2..) (Ex: BLASOLD, BLASNEW, SUBGRID, CORRECT)
  - Spatial maximum or minimum, possibly relative to a threshold (MAX (var, 0): Maximum of positive values).
  - Specific algebraic transformations
    - Celsius <-> Kelvin
    - Degree <-> Radian
  - Indexing operations:
    - Mask: Only the points listed in index have meaningful data and the others are changed to missing (Ex: MASK)
    - Scatter: Scatters the model data onto the points listed in index
    - Gather: Gathers from the input data all the points listed in index
  - Spatial "collapse" operations collapse of any dimension or combination of dimensions by various -possibly weighted statistical operations (mean, max, min, etc.)
  - Subspace: Extraction of subspaces or hyper slabs in any combination of spatio-temporal or other dimension
  - Others

The Transformer may support the following grid types:

- Horizontally:
  - Cartesian, i.e. the location of each grid point is given as a 2D array (i, j)
  - Regular or irregular or stretched in longitude and in latitude
  - Regular in longitude for each parallel, but unstructured in latitude (e.g. "Reduced" atmospheric grid)
  - Unstructured in longitude and in latitude
  - Staggered
- Vertically:

- V1 - Reproduction of the same horizontal grid at different levels  
The same horizontal grid is reproduced at different vertical levels. Each level has its particular mask. The vertical levels can be:
  - V1-1: Given at regular or irregular depth or height levels (z coordinate).
  - V1-2G hybrid: First level follows the topography (atmosphere models) or the bathymetry (ocean models), last level follows an isobar (atmosphere) or the surface (ocean), transition in between.
  - V1-3: Given at regular or irregular isopycnal (density) levels (r coordinate).
- V2 - Different horizontal grids at different levels  
The horizontal grid is not reproduced at different vertical levels. The horizontal grid can be rotated, translated, or totally unstructured.
- Other grid characteristics:
  - Grid may have masked grid points.
  - Grid may have "holes" (i.e. they do not cover the whole sphere).
  - Grid may be global or regional.
  - Grid may have overlapping grid points.

Other requirements for the Transformer may be:

- To support scalar coupling data.
- To support vector coupling data in the standard spherical geographical coordinate system.
- To support vector coupling data in any set of local co-ordinate system.
- To support fields with undefined variables.
- To support source and target coupling domains that totally or partially overlap (e.g. global atmosphere with a regional ocean, regional model nested into global model, etc.)
- For basic remapping (1st order conservative), to be able to calculate automatically the remapping info -address and weights.
- To be able to give some statistics and diagnostics on the coupling fields (mean, max, min, etc.) (Ex: CHECKIN, CHECKOUT in Oasis).
- To support coupling fields which characteristics may change over time as simulation develops (grid, resolution, distribution...).
- To be able to save, at a user defined frequency, its restart data (e.g. time accumulated data).
- To understand some standard conventions of meta-data.
- Based on meta-data description, to perform compatibility checks between data produced by source component and data required by target component.
- Based on meta-data description, to recognize type of coupling data (flux, vector, scalar) and verify that the user's transformation choice is appropriate.
- Based on meta-data description, to perform automatically some basic transformations (units -e.g. Celsius to Kelvin, order of dimensions -e.g. source (x,y,z) -> target (z,y,x), etc.), even if not prescribed by the user.

## Design Options for the Transformer Location

The transformations can be divided into 3 types:

- *Point-wise transformation*: An operation that can be completed on each grid point without any external information, neither from the neighbouring grid points, neither from another model. Thus they can be done locally without geographical knowledge and with any domain decomposition, such as time averaging.
- *Local transformation*: An operation that can be completed in a model without any information from another model, such as finding the maximum value of a field.
- *Non-local transformation*: An operation that requires information from another model, such as interpolation.

To perform these transformations, the Transformer needs information coming from the models (e.g. field units, grid, partitioning, etc.), and information prescribed by the user (e.g. the nature of the transformations). In a *static* coupled simulation, the information may be transferred only once initially; in a *dynamic* or *interactive* coupled simulation, these transfers must be possible at any point during the simulation.

The minimal option is that all transformations are necessarily performed in a Transformer entity distinct from the component model processes, as it is the case for the OASIS coupler today.

A desirable option is that at least point-wise transformations are performed locally on the component model processes by transformation routines included in the PSMILe, before any external exchange. This should be reasonably easy to achieve as it does not require any parallelization of the transformation routines, and is recommended in some cases to avoid extra communication. All other transformations are performed in a separate Transformer entity.

Another option is that point-wise and local transformations are performed locally in the component model PSMILe before any sending or after receiving the coupling fields. All non-local transformations are performed in a separate Transformer entity.

## Design Options for the Transformer Parallelization

In the case the option of performing point-wise and local transformations directly in the PSMILe is chosen, the full parallelization of the local transformation routines is required, as the PSMILe will in some cases be linked to fully parallel component models.

For the separate Transformer entity performing non-local transformations, the following parallelization options are possible:

### **Pseudo parallelization:**

Options for distributing the work of the separate Transformer entity, simpler than its full parallelization, could be followed as a first step. These options are however not scalable. In particular, they can lead to a waste of resources if the different data exchanges between any two models occur not simultaneously.

- Two-component-per-two-component approach

One separate sequential Transformer process is used for each pair of coupled components:

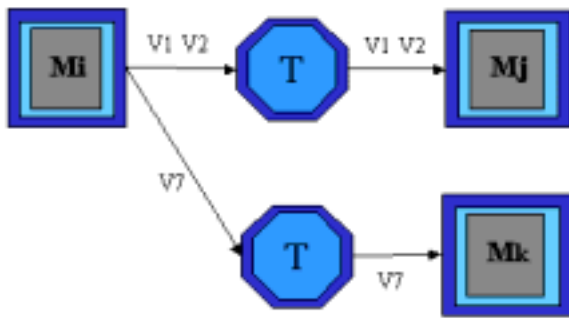


Figure 11: Two per two approach

This option is simple to implement, but presents the following disadvantages:

- The load balancing of the different separate Transformer processes cannot be ensured;
- Each model information has to be duplicated in the different Transformer processes.
  - Field-per-field bi-directional approach:

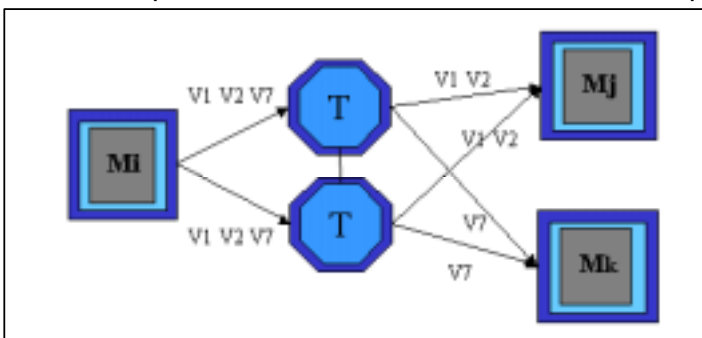
Between any two models, there are more than one separate sequential Transformer processes, each one treating an equal number of fields exchanged in both directions between any two models. This approach may ensure a better load balance but implies that each process has to calculate and store the information required for the transformation, e.g. the interpolation matrix of weights and addresses, in both directions. Furthermore, it also implies that each model- information has to be duplicated in the different related Transformer processes.

- Field-per-field unidirectional approach:

Between any two models, there are two separate sequential Transformer processes, each one treating the coupling fields exchanged in one direction. With this approach, each process calculates and stores only the information to do the transformation in one direction; this advantage is less significant if the transformation information in one direction can be automatically deduced from the one in the other direction. The main disadvantage is that the load balance between the separate Transformer processes can no longer be ensured.

- One executable full parallelization:

The full parallelization of the separate Transformer entity is mandatory if one separate Transformer executable performs all non-local transformations required between all components; the computing



load of this separate Transformer executable will be important and it is therefore important that it is parallel and scalable. (Of course, direct communication between any two components for which no transformations are required, with or without repartitioning, should still be possible.)

One **advantage** of this option is that the information about each model, for example the grid, is duplicated only once, namely in this separate parallel Trans-

Figure 12: One executable full parallelization

former executable memory. Another advantage is that it ensures an efficient use of these Transformer processes, especially if the different data exchanges between any two models occur not simultaneously. Furthermore, if the different coupling fields are available at the same time in the models, they can be packed into one message; this would present the advantage of reducing the number of messages exchanged and increasing their individual size.

One **disadvantage** is that the partitioning of this additional parallel executable can match the partitioning of only one model; repartitioning is therefore required for the other model components.

### **Many executables full parallelization**

If the repartitioning required for the above option proves to be too expensive, the option of using one additional parallel Transformer executable between any two component models could be envisaged; the partitioning of each additional parallel Transformer executable could match the partitioning of one of the two models.

### *The PRISM System Model Interface and Data-Exchange Library: Requirements and Design Options*

The PRISM System Model Interface Library (PSMILe) is the set of routines implemented in the model code to interface it with the rest of the PRISM System (other component models or additional coupling processes).

### **PSMILe general requirements**

The possible requirements specific to the PSMILe are the following:

- The modifications to implement in the model code are as reduced as possible.
- The PSMILe is layered, and complexity is hidden from the component code.
- The PSMILe includes the Data Exchange library as the most external layer.
- The PSMILe may include transformation routines if transformations are required locally before the exchange with the rest of the PRISM System.
- The component models are able to run in a stand-alone model without modifications, with or without an external Driver.
- The model information (e.g. length of a time step) is given by the model through the PSMILe and not duplicated externally by the user; this information may change during the simulation.
- The description of the data, i.e. the meta-data (e.g. units, grid coordinates, mask, length of a time step, distribution, ...), is given by the model through the PSMILe and not duplicated externally by the user; this information may change during the simulation.
- A good trade-off is chosen between (I) a concise list of parameters for each subroutine call (more subroutines provided with a shorter list of parameters) and (II) a small number of subroutines, each one having a longer and more complex parameter list. The complexity arises from the need to transfer not only the coupling data but also the meta-data.
- Interface routines once defined and implemented are not subject to modifications between the different versions of the PRISM coupler. However, new routines may be added.

- The PSMILe is extendable to new types of coupling data (e.g. data given on arbitrary grids).

### **Data Exchange Library requirements**

The Data Exchange library (DEL) performs the exchanges of coupling data between the component models, or between the component models and the separate Transformer entity. The DEL must therefore be included as the most external layer in the PSMILe.

The possible characteristics of the coupling data exchanges are:

- "End-point" data exchange: When producing coupling data, the source model does not know what other model will consume it; when asking for coupling data a target model does not know what other model produces it.
- The coupling data can be of different types: Integer, real, character, 1D-2D-3D-xD arrays, structures, operators, functions, ...).
- The coupling data are exchanged but also possibly on their meta-data, i.e. the description of the data (e.g. units, grid coordinates, mask, distribution, ...).
- The coupling fields characteristics, and therefore the associated meta-data, may change over time as the simulation develops (grid, resolution, ...)
- Coupling data produced by a source model can be consumed by more than one target model.
- Coupling data produced by one model may be only partially consumed by the target model; extraction of subspaces, hyper slabs or indexed grid points may be required before the exchange.
- Different coupling data produced by one model may have to be combined before the exchange.
- Algebraic operations may have to be performed on the coupling data before the exchange.
- The target model can consume the coupling data, produced by a source model, at different frequencies (i.e. one "put" will not necessarily match one "get" –time, integration / interpolation will be required).
- The occurrence of the exchange can be different for the different coupling fields.
- Occurrence of exchange is flexible (exchange can occur at a fixed frequency, fixed time intervals at different pre-defined time-steps, on given dates of a physical calendar -Julian, Gregorian, ...-, etc.).
- Coupling data produced from one model at a particular time may be required as input coupling data for another model at another time.
- The user does not necessarily define the Occurrence of the exchange initially; it can depend on parameters dynamically calculated during the simulation (conditional occurrence).
- Exchange points can be placed anywhere in the source and target code and possibly at different location for the different coupling fields.
- The exchange can occur directly between two component models without going through additional coupling processes.

- When the component models are parallel and have different data partitioning, repartitioning associated to direct communication is required; all type of distributions usually used in model component codes are supported. In a *static* coupled simulation, the characteristics of the repartitioning required between any two component-models are fixed, while in a *PM* or *CF dynamic* coupled simulation, they may change during the simulation.

Other specific requirements are:

- **Data exchange implementation:**  
The DEL offers efficient data exchange implementations for coupling models assembled into the same executable as well as for those assembled into different ones.
- **I/O and access to data files:**  
In some cases, input coupling data will not be provided by another model but should be read into a file indicated by the user in the coupling configuration file. This should be transparent for the component model and managed automatically.  
The format of these data files could be a standard PRISM fixed format. At a later stage, different formats could be supported for these data files; this would imply that the instance reading the file can interpret their content.  
For parallel component models, the I/O library will have to address the parallel I/O issue. One option is simply to avoid parallel I/O by doing regional selection for input data and by doing post processing operation after a simulation to recombine multiple output files provided by the parallel execution. MPI-IO is another option. Finally a third option is to set up a dummy application or I/O demon, which just acts as data source by reading the file and behaves like a regular model with respect to the coupled system. This last option is particularly interesting when the data present in the file need transformation, interpolation or repartitioning before being used by the model, and therefore is particularly interesting for parallel models. It is also interesting from the performance point of view if the I/O demon can perform the access to disk concurrently with the model calculations. However, it supposes that a Driver and an external I/O demon are active even for a component model running in a totally uncoupled mode.
- **Matching of output and input coupling data from different component models:**  
As discussed above, the DEL could perform, for static simulations, the matching between output coupling data produced by one model and input coupling data requested by another model. The matching may be based on the user's choices indicated in the configuration file, or may be done automatically when there is only one matching possibility. For dynamic simulations, information coming from the Driver is required.

### **An Important Design Option: A Common Model Interface Library for I/O and Coupling Data**

I/O and coupling data present many common characteristics and should therefore, in principle, share a common Model Interface Library. It should be evaluated further if this ideal concept can in fact be implemented without too many constraints.

The following list of characteristics shared by both I/O and coupling data was established:

- Data requested or made available by a model. Some data may be I/O and coupling data at the same time.
- Effectively, the model will not deliver all necessary data. For each particular simulation, the user has to activate some of them externally through a configuration file created with a GUI or any other means.



- Data, for which the "end-point data exchange" principle is applicable: The model itself does not know where the data come from or where they go. The user defines for each particular simulation the source/target models (for coupling data) or the source/target files (for I/O data) externally.
- Data, for which transformations may be required: The user prescribes these transformations externally.
- Some data required from another model in the coupled mode may in fact be forcing data read directly from files. In that case, the coupling library is faced with the same parallel I/O and meta-data interpretation difficulties.

The following list of differences was established:

- The number of coupling fields is generally smaller than the number of diagnostic output fields.
- One PRISM objective is to define a standard physical coupling interface between any two components, i.e. the nature of the coupling fields exchanged; standardization of the nature of the diagnostic output will be much more limited.
- Some local transformations required for I/O may not be required for coupling, and vice versa.
- I/O may require more or different meta-data to be transferred from the model.
- I/O data needs some mechanism to translate meta-data given by the model into CF-style description. This is required for coupling data only if the coupler is asked to generate its own coupling diagnostics files.

### II.2.3 Coupler review

This section surveys existing couplers or coupling applications, targeted or not to climate:

- OASIS from CERFACS
- Palm from CERFACS
- MpCCI from FhG-SCAI
- Calcium from EDF
- CCSM Coupler 6 from NCAR
- Distributed Data Broker from UCLA
- Flexible Modeling System from GFDL
- Coumehy from LTHE and IDRIS

For additional information on projects targeting or involving coupling aspects, on potential underpinning technologies, and on model developments related to coupling, the reader is invited to consult the document entitled "Met Office FLUME project - Model Coupling Review" ([http://www.metoffice.com/research/interproj/flume/1\\_d3\\_r8.pdf](http://www.metoffice.com/research/interproj/flume/1_d3_r8.pdf)), written by R. W. Ford and G. D. Riley from the University of Manchester.

## OASIS (<http://www.cerfacs.fr/globc/software/oasis/oasis.html>)

OASIS is the coupler developed at CERFACS, which will be the base of the PRISM coupler developments. It is primarily designed for coupling climate models.

The initial work on OASIS began in 1991 when the "Climate Modeling and Global Change" team at CERFACS was commissioned to build up a French Coupled Model from existing General Circulation Models (GCMs) developed independently by several laboratories (LODYC, CNRM, LMD). Quite clearly, the only way to answer these specifications was to create a very modular and flexible tool.

OASIS is a complete, self-consistent and portable set of Fortran 77, Fortran 90 and C routines. It can run on any usual target for scientific computing (IBM RS6000 and SPs, SPARCs, SGIs, CRAY series, Fujitsu VPP series, NEC SX series, COMPAQ, etc.). OASIS can couple any number of models and exchange an arbitrary number of fields between these models at possibly different coupling frequencies. The user defines in an input file 'namcouple' all the coupling parameters of the simulation for OASIS (models, coupling fields, coupling frequencies, etc.), namcouple is read at run-time by OASIS. Each component model of the coupled system remains a separate, possibly parallel, executable and is unchanged with respect to its own main options (like I/O or multitasking) compared to the uncoupled mode. OASIS handles only static simulations, in the sense that all component models are started from the beginning and run for the entire simulation. The models need to include few low-level coupling routines to deal with the export and import of coupling fields to/from OASIS.

The main tasks of OASIS are:

- Communication between the models:  
To exchange the coupling fields between the models and the coupler in a synchronized way, four different types of communication are included in OASIS. In the PIPE technique, named CRAY pipes are used for synchronization of the models and the coupling fields are written and read in simple binary files. In the CLIM technique, the synchronization and the transfer of the coupling message passing based on PVM 3.3 or MPI2 does data. In particular, this technique allows heterogeneous coupling. In the SIPC technique, using UNIX System V Inter Process Communication possibilities, the synchronization is ensured by semaphores and shared memory segments are used to exchange the coupling fields. The GMEM technique works similarly as the SIPC one but is based on the NEC global memory concept.
- Transformation and interpolation of the coupling fields:  
The fields given by one model to OASIS have to be processed and transformed so that they can be read and used directly by the receiving model. These transformations, or analyses, can be different for the different fields. First a pre-processing takes place which deals with rearranging the arrays according to OASIS convention, treating possible sea-land mismatch, and correcting the fields with external data if required. Then follows the interpolation of the fields required to go from one model grid to the other model grid. Many interpolation schemes are available: nearest neighbour, bilinear, bicubic, mesh averaging, Gaussian. Additional transformations ensuring for example field conservation occur afterwards if required. Finally, the post processing puts the fields into the receiving model format.

## *Palm ([http://www.cerfacs.fr/globc/PALM\\_WEB/](http://www.cerfacs.fr/globc/PALM_WEB/))*

The PALM project, currently underway at CERFACS, aims to provide a coupler allowing a modular implementation of a data assimilation system. In this system, a data assimilation algorithm is split up into elementary "units" such as the observation operator, the computation of the correlation matrix of observational errors, the forecast model, etc. PALM ensures the synchronization of the units and drives the communication of the fields exchanged by the units and performs elementary algebra if required.

This goal has to be achieved without a significant loss of performances if compared to a standard data assimilation implementation. It is therefore necessary to design the PALM software in view of the following objectives and constraints:

- **Modularity:** PALM provides a mechanism for synchronization of pre-defined functional units that can be executed in sequence, in concurrence, or in a mix of these two modes. One key aspect of PALM is also that dynamic execution of the units (i.e. units can be launched and stopped at any point during the simulation) or conditional executions of the units are allowed. PALM also performs the required exchange of information between these units.
- **Portability:** PALM aims to run on all the existing high-performance platforms and, if possible, on the next generation super-computers. This effort of "clairvoyance" can be accomplished only through the adoption of standard.
- **Performance:** PALM will be used in two modes: research and operational. The research mode will be used for the design of new algorithms and will prioritize the flexibility; on the contrary, the operational mode will work with a fixed configuration of the algorithm and will prioritize the performance optimization and the monitoring of the process.

PALM is a very flexible and efficient, but somewhat complex tool. For PRISM, it remains to be evaluated if this flexibility, and associated complexity, is required for coupled climate modeling.

## *MpCCI (<http://www.mpcci.org/>)*

The Mesh based parallel Code Coupling Interface (MpCCI) is a coupler written for multidisciplinary applications by the Fraunhofer Gesellschaft Institute for Algorithms and Scientific Computing (FhG SCAI). MpCCI enables different industrial users and code owners to combine different simulation tools. MpCCI can be used for a variety of coupled applications like fluid-structure, fluid-fluid, structure-thermo, fluid-acoustics-vibration, but was not explicitly designed for geophysical applications.

MpCCI is based on COCOLIB developed during the CIPAR project, funded by European Commission, and on GRISSLi-CI developed during the GRISSLi project, funded by the German Federal Ministry for Education and Research. MpCCI is not an open source product, but the compiled library offering basic functionality can be downloaded for free from the web site; special agreements apply for additional features like e.g. more sophisticated interpolation schemes.

MpCCI is written in C++ and is based on MPI1. MpCCI is mainly a parallel model interface library that provides the usual coupling functionality: 1- the interpolation of the coupling fields (including the neighbourhood search and calculation of weights and addresses), and 2- the exchange of coupling data between the codes (including data repartitioning when required). MpCCI also consists of a separate control process, which performs only a simple monitoring of the different codes, as MpCCI handles only static couplings.

Placing MPI like sending and receiving instructions in the coupled codes performs the coupling. MpCCI does not fully adhere to the principles of "end-point data exchanges" as each model has to know the target/source of its sending/receiving instructions. However, each code simply works with its own local mesh and needs no specific knowledge of the other code characteristics.

As MpCCI is based on MPI1, heterogeneous computing is supported as long as the MPI1 implementations of the different platform implied in the coupling allows it.

*Calcium* (<http://www.irisa.fr/orap/Publications/Forum8/berthou.pdf>)

Calcium is a coupler of codes developed by Electricite De France (EDF), and written, as MpCCI, for multidisciplinary applications. Calcium ensures the exchanges of coupling data between the codes in a synchronized way. The exchanges are based on PVM and heterogeneous coupling is allowed. Furthermore, Calcium automatically performs the temporal interpolation of the coupling data when the sending frequency of the source code does not match the receiving frequency of the target code. Calcium is used by about 10 different research or industrial groups mainly in France and is implemented in about 20 codes.

*CCSM Coupler 6* (<http://www.cesm.ucar.edu/models/cpl6>)

The Next Generation Coupler (NGC) - also called cpl6 - is the coupler being currently developed at NCAR, for the next version of the Community Climate System Model (CCSM), in the framework of the Accelerated Climate Prediction Initiative (ACPI) Avant Garde Project. They have performed the requirement capture, have described a design and are presently in the development phase.

The main characteristics of the NGCc are:

- The coupler is written in F90 and explicitly designed to couple four models: atmosphere, land, ocean, and sea ice. No flexibility concerning the number of models or their nature is included. Due to the nature of these components, exchange of 2D fields only is supported. These four models can run concurrently, sequentially or in a mix of these two strategies. Each component and the coupler are separate executables.
- The coupler can run in parallel decomposed into an arbitrary number of processors, and supports the following type of parallelism: pure shared-memory, pure message-passing, and hybrid parallelism incorporating threading on multiprocessor nodes and message passing between the nodes.
- The transformations performed by the coupler include interpolation (conservative remapping using the SCRIP library), merging of coupling data originating from multiple source grids, time accumulation and time averaging, diagnostic computing, and writing of history data sets, and also computing of certain interfacial fluxes between components. This choice was made as the fluxes need to be calculated at the higher resolution AND at the higher required frequency, which may be characteristics belonging to different models.
- All coupling data exchanges are performed with MPI. Parallel exchange of coupling fields and repartitioning is possible. However, all coupling field exchanged between any two components have to go through the coupler where the transformations are performed; direct component to component exchanges are not allowed.

One can note here that the goal of achieving efficient vector processing performance was not identified as a mandatory requirement for the NGC.

## *UCLA Distributed Data Broker (<http://www.atmos.ucla.edu/~drummond/DDB/>)*

The Distributed Data Broker (DDB) is a software tool designed to handle distributed data exchanges between the UCLA Earth System Model (ESM) components. These components are: Atmosphere General Circulation Model, Atmospheric Chemistry Model, Ocean General Circulation Model, Ocean Chemistry Model, and are run as separate executables. The DDB is composed of the Registration Broker (RB) and of three libraries linked to the component models: the Model Communication Library (MCL), the Communication Library (CL), and the Data Translation Library (DTL).

The Registration Broker is a process that collects model information from the models initially. The RB is only active at the beginning of the coupled run; thus, any model process implied in the coupling can take this role and later resume with normal model operation. The DDB follows the "end-point data exchange" principle, which they call the producer-consumer paradigm. In the registration phase, the different models register their "production" and "consumption" of coupling data and RB performs the appropriate matching which will be effective at run-time.

The MCL contains a set of callable routines that are used by the different component models to register at the beginning of a run, and perform the exchanges of data at run-time.

The CL is a set of routines used by the DDB to manage data exchanges based on the communication libraries available on the computer platforms. At run-time, the model producing the data sends the data directly to the consuming model at a given time interval; the consuming model will later receive the data at a rate dictated by its internal computations. Heterogeneous coupling is allowed as long as the communication libraries available on the computer platforms support it.

The DTL transforms data in a given grid domain to the domain of the requesting model. This library will include routines ranging from simple linear interpolation to high order data translation routines.

## *GFDL Flexible Modeling System (FMS) (<http://www.gfdl.noaa.gov/~fms/>)*

The design of the FMS is geared toward coupled climate models running as a single executable. The component models that can be included into the FMS are atmosphere, land, ocean and ice models. In the FMS, the coupler is the main program, which drives the components models. To interact, these components communicate only with the coupler. They may be on different grids and have different data decompositions and the coupler manages the transformations required between them. Recently, some parallelization concepts were experimented on the component models themselves, using abstract parallel dynamical kernels: the parallelism is in fact built into basic operators invoked in the model, including arithmetic operators as well as differential operators such as curl, div, grad and laplacian.

## *COUMEHY (contact: C. Messenger, [messenger@hmg.inpg.fr](mailto:messenger@hmg.inpg.fr))*

COUMEHY is an ongoing French coupling project, involving the "Laboratoire des Transferts en Hydrologie et Environnement", "Hydrosciences Montpellier", the MEVHySA team from the "Institut de Recherche pour le Développement" from Montpellier, and the "Institut du Développement et des Ressources en Informatique Scientifique". The objective of this scientific and technical project is to couple one atmospheric model to different hydrological models running on different platforms, in order to evaluate the importance of coupling processes between atmospheric and continental hydrological cycles, in the climate global change perspective. The inter-operability of different codes

running on different machines was in that case a strong requirement: the choice was therefore made to base the communication on CORBA (Common Object Request Broker Architecture)

## II.3 Data Management, Analysis, Visualization and Archiving

*This part of the PRISM System Specification details the High Level component structure of the data management system, which integrates the data management, analysis and visualization with overlapping functionality (coupling in particular).*

### II.3.1 Processing Library

PRISM aims to provide software that will do both processing in bulk (e.g. for post-processing of model output files) and computations on smaller amounts of data for analysis and visualization. When developing the processing library we will share code with the I/O library and the coupler.

### II.3.2 Visualisation

Name	Description
OpenDX	OpenDX is a uniquely powerful, full featured software package for the visualization of scientific, engineering and analytical data: Its open system design is built on a standard interface environment. And its sophisticated data model provides users with great flexibility in creating visualization. Is Open Source.
Vis5D+	A free OpenGL-based volumetric Visualization program for scientific datasets in 3+ dimensions.
Vis5AD	VisAD is a Java component library for interactive and collaborative visualization and analysis of numerical data
Ferret	Ferret is an interactive computer visualization and analysis environment designed to meet the needs of oceanographers and meteorologists analysing large and complex gridded datasets. Interactive command-line and scripting language.
GrADS (Grid Analysis and Display System)	Interactive desktop tool that is used for access, manipulation and visualization of earth science data.
NCAR Graphics	A time-tested UNIX package, consisting mainly of over two dozen Fortran/C utilities for drawing contours, maps, vectors, streamlines, weather maps, surfaces, histograms, X/Y plots, annotations, and more.
GMT (General Mapping Tools)	GMT is a collection of public-domain Unix tools that allows you to manipulate x,y and x,y,z data sets (filtering, trend fitting, gridding, projecting, etc.) and produce PostScript illustrations ranging from simple x-y plots, via contour maps, to artificially illuminated surfaces and 3-d perspective views in black/white or 24bit colour.
COCO library	A Python library with C/C++ underneath, using some modules from CDMS, to implement objects in memory containing data and CF-based meta-data. I/O to netCDF or PP. A more systematic specification for the high-level operations than the UKMO PV-WAVE library.
CDAT (Climate Data Analysis Tool)	Includes Climate Data Management System (CDMS). Object-Oriented data management and analysis system, whose user interface is in Python. Open source. Extensible by contributing Python modules. Interface to VCS (visualization).
IDL (Interactive Data Language)	Commercial Software for data analysis, visualization, and cross-platform application development
AVS (Advanced Visual Systems)	Commercial Software.

Table 14: Graphics packages considered for PRISM

The Table 14 lists existing freely available packages, with the exception of the commercial software IDL and AVS, which are in use for climate data analysis and hence should be considered for use in PRISM. Each of the packages will be evaluated; the results of which will be presented in ARCDI.

### II.3.3 High Level Architecture

The following diagram depicts the components comprising the Archive, Data Processing and Visualisation system and illustrates how they fit together. The Job Flow and Run Shell show the coupled model system running forward in time. Some data processing, such as time-step averaging and area selection, can take place within the run shell. Data is then output from the model to files via the I/O layer. These are then picked up and sent to the Run Shell Data Processing component, which will post process the data further, performing tasks such as climate meaning etc. From here, the data is sent to the Data Repository (an active archive) through a common Archive Interface that will be defined within PRISM. Located under this interface are site-specific procedures to prepare and send the data to the archive.

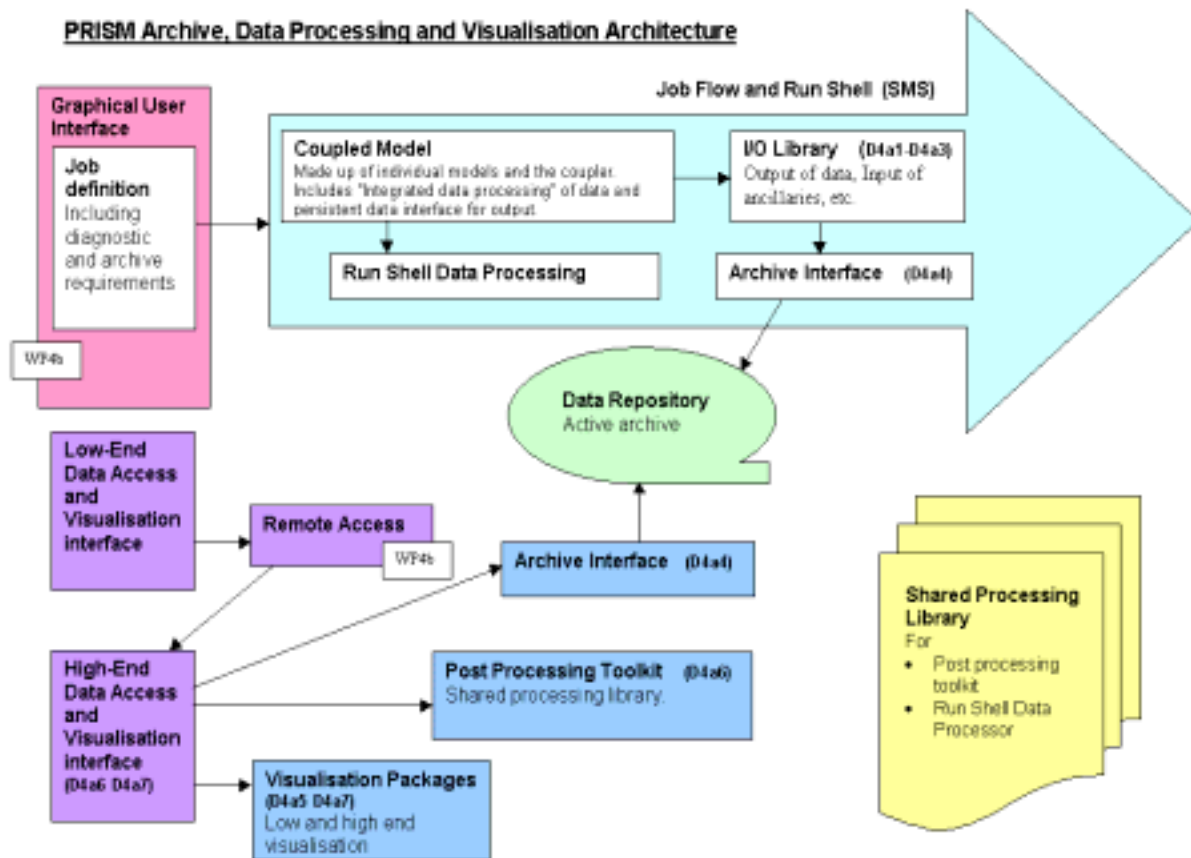


Figure 13: Diagram of the archive, Data Processing and Visualisation System



All these components are glued together by software in the Job Flow and Run Shell (an SMS like system to perform job control) and initiated through a user interface. The current design enables all this to run at one site, with the possibility for the user interface client and server to be run remotely.

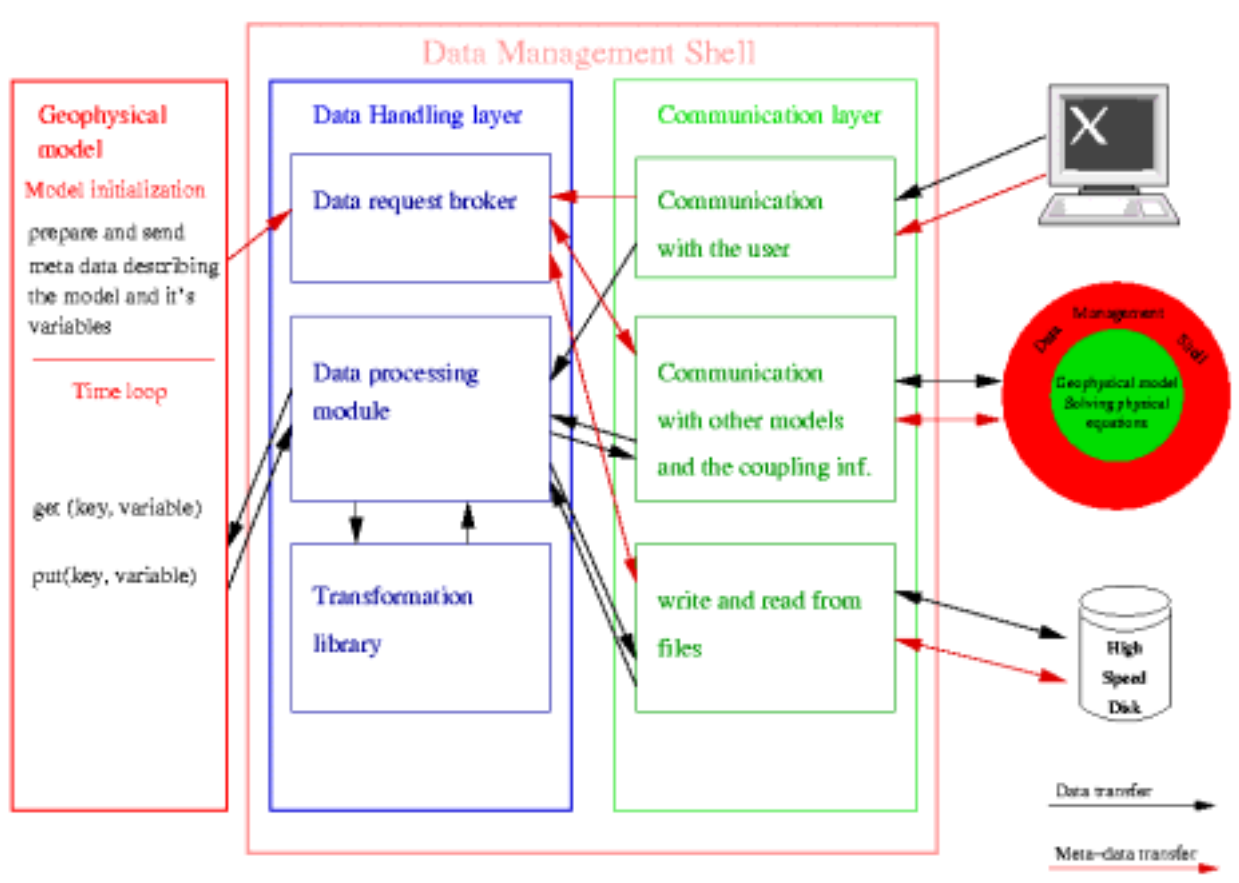


Figure 14: Internal Data Management structure for the Job Flow an Run Shell

A user interface will be provided to access data at a given (potentially remote) site. Once again, the archive interface will act as a common gateway for all PRISM software with site specific code layered beneath this. Data is filtered and either processed remotely (low-end visualization) or locally (high-end visualization) before being presented for display.

Where possible, software will be shared between components. Such piece of software is the Shared Processing library, which needs to be available on all platforms.

The diagram on the left illustrates the Internal Data Management structure for the Job Flow and Run Shell.



## II.4 PRISM user interface

*Design options and constraints of the user interface (UI) are described. The UI interfaces the system to the user. The user should be able to use the UI to access resources remotely, i.e. the users do not have to be physically in the same place as where the model is executed or data is located. To achieve this remote access, a web-based system is recommended, since it allows access to any resources via the HTTP protocol. This is currently the only recommended protocol on the Internet, which is accepted almost everywhere. While there is a benefit to have a command line interface for low speed connections, complex configuration tasks are best accomplished with a graphical user interface (GUI) providing visual support and guidance. The detailed functionality of the UI is specified.*

---

### II.4.1 Design Options and Constraints

#### *Constraints*

A graphical user interface (GUI) should implement a high level of visual guidance. An integrated documentation environment and online help should be provided.

The GUI should allow the users to use both keyboard and mouse navigation.

Each function of the GUI should be defined such, that it can be implemented as a separate module by a responsible subgroup. However, this may result in the use of different tools for different aspects of the UI. All functions managing the overall experiment structure or each individual model (coupler) component or task must specify the integration level with the GUI.

The GUI should allow to import/export setups from/to other users.

The GUI should let users configure the layout of the system, such as colours and fonts used.

The GUI should run even when the PRISM remote application server is unavailable and allow the user to create setups for later submission.

#### *Options*

It is important to specify the integration level of the different functions and the interaction style (design of the behaviour and the look-and-feel of the UI). The use of existing products and their design constraints have to be reviewed in this context.

Client software is often described as "thin" or "fat", referring to the functionality available in the client without the need for calling on the help of the server. The fat client allows for a high degree of programmable support and sophistication as well as making good use of the clients computing facilities. The latter can be an advantage or disadvantage depending on the device/computer in question. The larger size of the client application for a web application means longer initial start-up time. A thin client will make use of the server often and will run in some hosting application such as a web browser. The possibility of providing sophisticated applications is limited and dependent on

the capabilities of the hosting application. Thin clients are fast to start and can be run on many devices but are dependent on available network speed and the power of the cooperating server.

The use of a Java client is the best choice as it enables the client to provide more built in support, recover from server failures, execute in a standard environment and be less sensitive to browser differences. A desirable option is to be able to configure the functionality of the client to make it more light-weight .

Elaborate authentication procedures are annoying to users. Special consideration should be taken to allow the security mechanisms of the GUI to be shared by added software components. Therefore, it is important to present a single authentication/login interface to the user.

## II.4.2 Commercial Components

Any licensing or usage restrictions for any tool specified in the requirements needs to be evaluated.

## II.4.3 Hardware/Platform Interfaces

The GUI should be implemented as platform independent as possible. Java represents the only candidate available that can be deployed over multiple platforms as an application or incorporated into a mainstream Internet browser. For reasons mentioned earlier the use of a thin client (HTML) limits the scope of intelligence in the tool.

## II.4.4 Software Interfaces

This should describe the relationship between the different functions of the GUI, i.e. visualization, configuration, archiving and monitoring. A programmatic API must be specified to allow new functions to be integrated into the GUI. This API specifies functions to allow new applications to be called and to extract information from the GUI regarding experiments. The remote communication capabilities of the GUI should be available for software that integrates with the GUI.

## II.4.5 Communication Interfaces

As this is a web oriented UI all communication is expected to be in the HTTP protocol to allow it to pass through the firewall of different sites.

## II.4.6 Performance

The performance of a GUI felt by the user will vary considerably depending on the characteristics of the users hardware but will also vary between different software implementations from different manufacturers. The factors relating to the performance are mainly:

- Network speed - The slower the network link the longer it will take to transfer the program to your local workstation and to execute command that communicate with the remote server. The network speed over Internet is expected to improve as technology advances.

- Memory - The available memory on the client will influence how much user data (experiments or their results) can be worked on at any one time. It will also limit the functionality that can be implemented in the client.
- CPU performance - Once the GUI application has started it will depend on the CPU-power of your local hardware.

As most of the factors outlined above are out of the developers control there needs to be a specification of a minimum requirement with respect to the client hardware to make sure all clients can use the GUI. This includes the size of the display. The performance factors may exclude doing visualizations or other performance stressing tasks in the GUI application.

## II.4.7 Extensibility

The UI should be extendable by users. The level of extensibility and customization needs to be defined. Envisaged future extensions need to be considered.



## III - PRISM System Constraints

### III.1 Review of Component Models

*A technical review of existing component models is given. The list of models gives the state of the model review up to June 2002, it is not closed but can be extended in the future.*

---

#### III.1.1 List of Models

##### Atmosphere

- ARPEGE-Climate - Action de Recherche Petite Echelle Grande Echelle
- ECHAM5 - Max Planck Institute for Meteorology Climate Model
- GME - German Weather Service Climate Model
- HIRHAM - Regional Model for the atmosphere
- LMDZ - LMD/IPSL Atmospheric general circulation model
- RACMO - Regional Atmospheric Climate Model
- RCA - Rossby Centre Atmosphere Model
- UMA - Unified Model - Atmosphere Component

##### *Atmospheric Chemistry*

- INCA - INteractions with Chemistry and Aerosols
- KNMI\_TM - Atmospheric Chemistry Transport Model
- MOZART - Model of ozone and related tracers

##### *Land-Surface Schemes*

- ISBA - Interaction between Soil, Biosphere and Atmosphere
- MOSES – UK
- Met Office Land surface model
- ORCHIDEE - ORganizing Carbon and Hydrology In Dynamic EcosystEms
- RCA-soil - Rossby Centre Land-surface scheme

##### *Ocean*

- C-HOPE - Hamburg Ocean Primitive Equation Model
- OPA - Océan Paralléllisé

- MICOM - Miami Isopycnal Coordinate Ocean Model
- MOM - Modular Ocean Model
- HYCOM - HYbrid Coordinate Ocean Model
- RCO - Rossby Centre Ocean Model
- UMO - Unified Model - Ocean Component

### *Sea-Ice*

- LIM - Louvain-la-Neuve Sea-Ice Model
- NERSC Sea-Ice Model
- RCI - Rossby Centre Sea-Ice Model
- UMI - Unified Model - Sea-Ice Component

### *Ocean-Bio-Geochemistry*

- HADOCC - Hadley Center Ocean Carbon Cycle
- PISCES
- HAMOCC - HAMBURG Ocean Carbon Cycle

## III.1.2 Technical Aspects

### *Global models*

#### **Coupling**

All component models listed above have already been used in coupled configurations, using different coupling approaches. For Ocean-Atmosphere coupling the data exchange is handled in most cases by the coupling software OASIS. Examples are the coupling between OPA and ARPEGE or C-HOPE and ECHAM5. Ocean and sea ice models belong in most cases to the same executable. Coupling data are passed as arguments of Fortran subroutines or belong to COMMON data blocks. The OASIS coupler is used mainly for data exchange with the atmosphere components (ARPEGE and LMDZ). In all cases, the biogeochemistry is a package of the calling ocean model, since biogeochemical tracers are calculated on the same grid as the ocean prognostic variables. In all current implementations, the land surface scheme is a package of the atmosphere model. Some of the land surface schemes are closely adapted to the atmospheric components like in SPA - ARPEGE - Climate or in RCA - RCA - soil. Some components (e.g. the Unified Model) exchange data through COMMON data blocks or the data are written to an output file. The method chosen depends on the user's choice and on memory constraints.

#### **Input/Output**

Currently models handle their output in many ways. Nevertheless a majority already supports the NetCDF output (ECHAM5, GME, LMDZ, HAMOCC, OPA, MOM, LIM, ORCHIDEE, PISCES, and



INCA), some of them through the IOIPSL library (LMDZ, OPA, LIM, ORCHIDEE, and INCA). For atmosphere models, GRIB is another common format. The Unified Model components output is written as binary data using Met Office pp-format, comprising a header plus the binary field itself.

### **Further technical remarks**

Most of the reviewed models are programmed either in Fortran 77 and/or Fortran 90 with small parts written in C. Only ECHAM5 requires Fortran 95. All model components that are able to run in parallel using 1-dim or 2-dim domain partitioning handle the data exchange between processes using the MPI1 standard.

### *Regional Models*

Regional climate model systems are investigated in a separate work package. The reviewed models are all used in regional climate studies, both in present-day and in scenario mode, in national projects and European ones. These systems have different backgrounds, represent different philosophies in how to construct a regional climate model, and have different links, due to historical and practical reasons, to particular global models. The SMHI/Rosby Centre model (RCA) is based on the HIRLAM operational limited area model, but with physical parameterizations modified/replaced with high-resolution climate applications in mind. The DMI (HIRHAM) model is based on HIRHAM dynamics, but its physical parameterizations are based on those of the ECHAM5 AGCM. The Hadley Centre regional models, being part of the UKMO Unified Model system, share in principle the parameterizations of the respective global models.

### **Coupling**

Similarly to the components used for global simulations described above, each of the three regional model systems have an atmospheric module which calls a (land) surface scheme as a subroutine. One of the systems includes sulphur chemistry. One of the regional model systems comes with a regional ocean - sea-ice module. In contrast, the atmosphere - land-surface module plus routing and the ocean - sea-ice modules can be run independently. So far the regional ocean model system has been applied to the Baltic Sea, but adaptation to the Arctic is being considered. The regional atmosphere-ocean coupling uses OASIS.

All aspects of the coupling algorithms are difficult to determine, because parts of the models are integrated within all-in-one codes. In principle, it should be possible to run most components in a "concurrent by nature" mode.

### **Input/Output**

All reviewed models (various atmosphere/land-surface models, and the ocean/ice models) have their own restart mechanism. Restart files have various formats. All components need parameter input from external data files. A restart can be done from a combination of input from a global model and climatological data, or from "true" restart fields generated during a previous integration.

The input required is generally in the form of 3D (for some variables, 2D) data in geographically limited domains, e.g. through lateral boundary relaxation zones, 4-10 grid points wide, at all sides of the regional domain, or the 2-D sea surface state when no regional ocean model is included. Typical source of input is global model data. Within PRISM, global data should be provided by a GCM (AGCM, OGCM or AOGCM) through the PRISM coupler in a standardized way. In addition, a

regional model component might require input from other regional components. The minimum coupling frequency for the global-regional coupling is at present  $O(6 \text{ h})$ . However, this is expected to increase in the future, up to the limit given by the two time-steps of a global and the regional model.

The most frequently used data format is GRIB.

Note that, regarding coupling regional models (such as atmosphere-ocean), problems and options are in principal the same as in global model systems.

Diagnostic output is controlled by different methods, ranging from modifying the code to using name-list input.

### **Further technical remarks**

So far, the various models have been applied to Europe, USA, Eastern Pacific, Africa, India and the Arctic. All models use staggered grids together with a variety of vertical coordinates: hybrid (sigma-geopotential) and geopotential coordinates. The PRISM system should ideally allow automatic configuring of a regional model anywhere on the globe.

The parallelization strategy is usually based on horizontal partitioning.

Overall, the regional models encompass a number of model components very different in detail, but with common requirements regarding the global-regional coupling. Requirements regarding internal coupling between the components in a regional model system are in principle the same as in global model systems.

## III.2 Review of Existing and Future Target Computer Platforms

The state, the trends, and the future of High Performance Computing (HPC) in relation to the development of PRISM components are assessed. This investigation is motivated by the need to early identify the best suited programming languages and paradigms as well as parallelization strategies targeting the reduction of porting and maintenance efforts of the software product over different platforms, as well as a good performance and scalability on leading edge HPC facilities over the years to come.

The present assessment of HPC technology trends does not aim to cover all aspects that one might associate with HPC. Rather, it is tuned to some of the specific needs of the PRISM Project, and its climate community. The following quotes from the influential "PITAC report" [Joy, Kennedy 1999] would like to underline the visions and challenges of this document: "Drive high-end computing research by trying to attain a sustained petaops/petaflops on real applications by 2010.." and, we would like to add, in particular for the different PRISM components.

### III.2.1 Evolution of the HPC Market

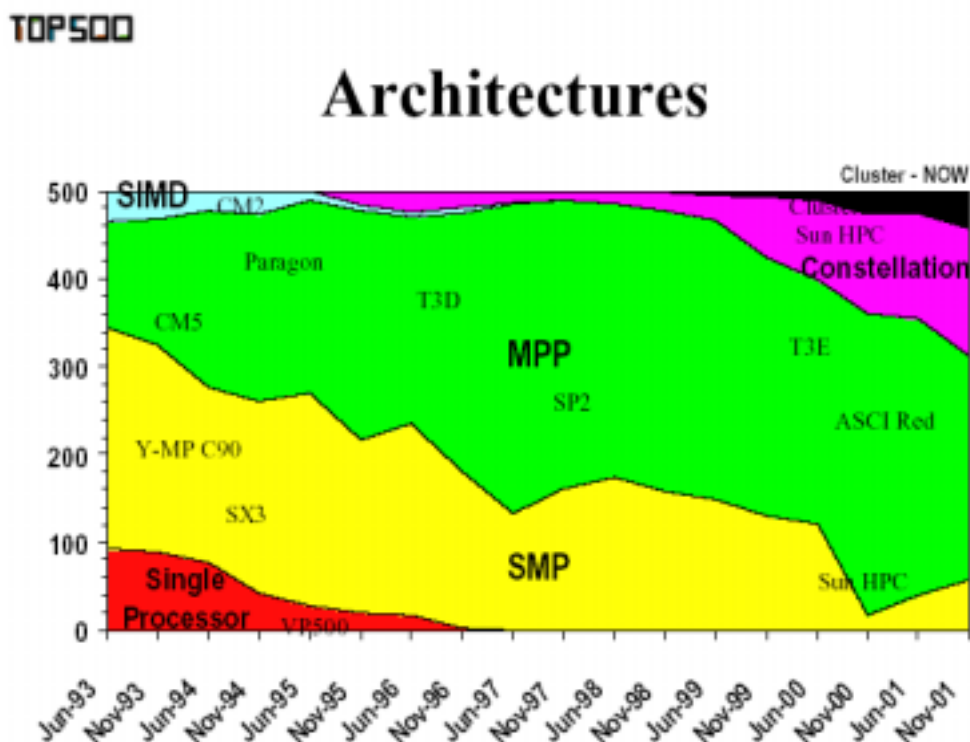


Figure 15: TOP 500-Architecture evolution over years.

A decade ago, the scenario looked a little bit different, chip technology was mainly dominated by proprietary developments, while today the trend is clearly driven by the main idea 're-using' the same technology from desktop workstations through servers up to high-end systems. This trend can be further extrapolated by predicting a convergence of chip technology into 3-4 main chip providers, often as a result of company mergers or high level commercial collaborations. Concerning the architectures, a decade ago high-end single processor, and single nodes SMPs were quite common, today the high sustained performance can only be reached via combination of both high single CPU performance and high degree of parallelization.

The real technical change over the past decade has happened around systems built on commodity building blocks (commodity in a broad sense, in this context an entire SMP server can be considered a 'basic building block' unit).

It is important to underline the central role played by the integrated HPC-solutions and the broad and complex spectrum of corresponding HPC-know-how, which mark the real distinction between different solutions (OS-related functionality and performance features for handling huge amounts of data, single system image related features, compilers, tools, libraries etc.). The many quality-related "HPC details" very often make the real difference between the available "clusters of clusters". The real value of such diverse integrated solutions can only be assessed on the basis of real-life benchmarks based on end-user applications and full cost considerations. Experience shows that permanent and pro-active benchmarking of a set of kernels representing present and future concrete needs of the user community against emerging HPC-solutions may provide the best basis for the preparation of a particular HPC-procurement. In this sense the effort of the WP2b is proposed to the PRISM community as a permanent service monitoring HPC-technology trends.

Figure 15, detailing the world's top 500 number crunching systems, documents the rise of massively parallel systems (MPP) and the start of the growth of network of workstations (NOW).

### III.2.2 HPC Architectures: State-of-the-Art and Trends

By extrapolating the trends of the past decades into the near future, with particular respect to base technologies and market trends, it appears safe to draw two conclusions. First, basic concepts underlying HPC, such as architectural features and trade-offs, are well-understood and are unlikely to change. Second, in terms of parameters such as speed, memory size, transfer rates, and cost, the only certainty is permanent improvement, but at different rates for different system components. Thus, systems design favours different balances at different times among various components such as processors, memories or communications. As a consequence, the products that dominate the market keep changing with time.

This section reviews some of the main arguments in support of the thesis that the HPC scene is changing:

A trend that seems to emerge is that most new systems look as minor variations on the same theme: clusters of RISC-based Symmetric Multi-Processing (SMP) nodes, which in turn are connected by a fast network. Culler et. al. in [Culler 1998] consider this as a natural architectural evolution. However, it may also be argued that the requirements formulated in the ASCI program, as well as pure market volume of HPC systems in general, has steered these systems in this direction.

This trend is amply documented in the literature, e.g. in the "Top 500"-list. Although this list, compiled every 6 months by the researchers Jack Dongarra at the University of Tennessee, Hans Meuer at the University of Mannheim and Erich Strohmaier at NERSC, is only one measure of per-

formance, it is the best known and most widely accepted one, simply because no other comparison of such a large number of powerful computers exists anywhere.

The Top 500 list is based on the Linpack Benchmark, solving a system of linear equations, perhaps the single mathematical operation typical of the broadest range of numerical computation. It measures computing power close to peak capacity, and thus is not necessarily representative of computations with irregular data access patterns. Other measures also have their limitations. Efficiency (sustained performance divided by peak performance), e.g., under-values machines, which are inefficient by this measure but cheap. While acknowledging the limitations of the Top 500 list and the Linpack Benchmark, the fact remains that no other published comparison comes close to defining the world of supercomputing as well as "Top 500". Although we are strongly against the idea to rank a system only based on this single benchmark, we have the feeling that this list is anyway representative of the HPC market, identifying at least the leading edge HPC systems and HPC manufacturer.

A word about the "bird's eye view" approach presented in this section: A detailed justification of the summary statements made could easily fill volumes. There are hundreds of scientific papers, in addition to trade journals and commercial documents, which assess various aspects of the current HPC scene, and attempt to extrapolate them into the future. The reference section lists some sources as entry points to the vast literature.

### *Architecture: Driving forces*

We identified at least three different forces that directly influence supercomputer technology trends.

- **Technology Progress in General**

Most experts agree that the exponential increase in processing speed and memory/storage density will continue for yet another decade (well beyond the planning horizon of this report). As a consequence, every supercomputer loses its competitive edge after two to three years of use. The much longer lifespan of application software makes software compatibility and portability a central issue in purchase decisions.

- **Market**

"The computer business has become a commodity business - with all that that implies. Thus, it will be driven by economic factors rather than technology..." [Preston 1998].

The number of supercomputers sold each year is on the order of a few thousand. In contrast, hundreds of millions of microprocessors for PCs, cellular phones and game stations are produced; millions of PCs and workstations based on these commodity items are produced. These numbers lead to large economies of scale, as design and non-recurring manufacturing costs are spread over a large number of chips, boards, and finished products. In contrast, for highly specialized high-end CPU (like vector CPUs), which only sells few hundred systems, for which only a few thousand processors are built, design costs easily dominate the overall cost.

Also the vector CPU has passed a very dynamic evolution since it has been successfully moved to the CMOS technology and can be realized today on a single chip. Thus the CPU architecture design can continuously evolve leaving room to real competitive advantages to the engineering "details" (like packaging, cooling and number of pins, etc.).

Moreover the appealing idea to reuse the same CPU technology from PC or workstations to high-end systems start becoming a reality for most HPC vendors.

- **National and Industry policies**

Since several decades, the US and Japanese governments are effectively subsidizing the

development of supercomputers by commissioning and buying ever more powerful systems. The most impressive examples are the US ASCI program and the Japanese Earth Simulator project, which are de facto HPCS technology driving forces [Various sources, 2002].

This new machine is five times as fast as the best of the top six supercomputers in the world (which all were in US) and is being used by a large number of research institutes in the field of earth sciences. A German supercomputer expert says it is likely to remain the world's fastest supercomputer for at least two years. Hans Werner Meuer, founder of the top 500 list of world supercomputers, says he expects the Earth Simulator to outperform all of its nearest 19 rivals put together.

The previous "Top 500" record holder, 'ASCI White' at the Lawrence Livermore Laboratory in the US, was built by IBM and is used to simulate nuclear weapons explosions. ASCI White has 8,192 processors, while the Earth Simulator has only 5,104.

This development strongly challenges the ASCI road-map by introducing new quality in reaching the highest sustained performance for real-life applications being the combination of the high level parallelization and the powerful and efficient (both from the sustained performance and energy/heat considerations point of view) CMOS based vector CPUs.

### *Architecture: Many Weak versus a Few Powerful Components*

All current supercomputers are parallel machines, with a dozen to ten thousand processors (the Top 500 list of November 2001 lists only few systems with fewer than 64 processors).

Although today's microprocessors are an order of magnitude slower than vector processors, they are much cheaper, and companies invest much more resources in their future development.

Several systems consisting of a few hundreds of commodity microprocessors have entered the "Top 500" list over the last decade. Interconnection technologies vary widely and will play a key role in the future of high-performance computing due to the general trend to build systems on higher and higher number of CPU's.

Still the main question is whether a computer consisting of a large number of cheap commodity components can deliver, for a given application, the same sustained performance than computer consisting of a small number of expensive, tailor-made components: what can be stated is that the equation is not as simple as it might be thought at first.

For a relatively small amount of CPUs, solutions based on cheap commodity components can be orders of magnitude cheaper than tailor-made components. But several other aspects start playing an important role when assembling systems made out of several thousand of CPUs targeting sustained performance in the order of TFlops sustained...

One of them is the well-known Amdahl's Law, defining how much parallelism a given application can usefully exploit. This issue can become a real nightmare for large MPP systems when also considering an additional aspect that dramatically increase as soon as the number of CPU become large: load imbalance. This consideration could be in favour of future systems with efficient specialized CPUs, where the targeted sustained performance can be achieved with a relatively small number of CPUs.

Other aspects are related to system stability (fault tolerance) system scalability (hardware as well as operating system), inter-node communication (latency, bandwidth) and I/O performance.

Finally it will always be the precise requirement of a given scientific problem to define on the basis of an application benchmark and the full costs considerations the optimal investment balance between the power of a single CPU and the number of CPUs.

## *Architecture: Taxonomy*

Since many years the taxonomy of [Flynn 1972] has proven to be useful for the classification of high-performance computers

- **SIMD (Single Instruction Multiple Data)**

Vector computers are an important subclass of these systems. They use the fastest processors currently available. Popularized by Cray in the seventies and eighties, they lost most of their market share in the nineties because the low production volume (partially due also to US trade embargo) led to high cost. The NEC SX-6 is a typical representative of this kind of computer.

- **MIMD (Multiple Instructions Multiple Data)**

These machines execute several instruction streams in parallel on different data. A further important subclass taxonomy of this kind of systems is related to memory accessibility.

- Shared memory systems: Shared memory systems have multiple CPUs all of which share the same address space.
- Distributed memory systems: In this case each CPU has its own associated memory. The CPUs are connected by some network and may exchange data between their respective memories when required. A few years ago, massively parallel processors (MPP) constructed of tightly coupled microprocessors were fashionable. Today, one trend goes toward systems based on a larger number of more loosely coupled PCs or workstations that require little development effort [Baker 2000].
- Hybrid systems, clusters of SMP: In this case a mixture of the previous two technologies is used. The idea is clustering together a large number of SMP nodes connected by specialized and dedicated fast network interconnection.

Further architectural issues include:

- **Memory**

On most of the current architectures memory is physically shared by only a small number of processors. In some cases it can be shared by even up to a several hundred processors, however, a coarse distinction has to be made by looking onto the memory hierarchy in order to achieve certain scalability. On symmetrical multiprocessor systems (SMP) any memory location can be reached ideally with the same latency and the same bandwidth from any CPU. Scaling this architecture in hardware up to high processors counts is a very expensive engineering effort. The Cray T90 with up to 32 CPUs on a flat memory was one of these representatives.

In order to scale the shared memory approach up to several hundred of processors the approach of a distributed shared memory (DSM) was implemented by several HPC architectures. DSM allows shared memory programming with a transparent memory coherent protocol. The memory coherent protocol might be implemented in hardware or software. DSM computers are characterized by a fast memory access within smaller units of a few CPUs and a slower one between these units. This scenario is often called non-uniform memory access (NUMA). Software development might be harder on DSM systems since the memory hierarchy has to be taken into account although tools are available to assist developers during software development on this kind of systems.

- **Interconnection**

A set of workstations can be connected to form a supercomputer either by using standard network technology (e.g. Fast Ethernet), or by custom-made high-performance interconnects. It seems that Fast Ethernet is adequate only for connecting a small number of nodes or for applications with little communication between processes.

An extensive, detailed and complete architecture description of most of the current HPC systems can be found in [van der Steen 2001].

## Software

Typically, neither end-users nor "experts" can readily figure out what the main structural features of a given program are. The choice of data structures, data layout and access pattern can affect performance by orders of magnitude. Yet software manuals rarely describe such important design and optimisation aspects, at least not in a legible and formal manner. Efficient software design for complex architectures is likely to remain one of the most difficult aspects of HPC.

A quote from [Cipra 1999]: "... advances in computer capabilities, which have tempted researchers to develop more complex programs and tackle larger problems, have made speed trickier than ever to achieve. Programs tailored to run fast on a particular machine may not work at all on others, while "portable" codes often run inefficiently on all platforms. Programmers expend a lot of effort fitting software to hardware - and the target keeps moving".

Writing software for supercomputers is a tedious task. To get high performance, the programmer has to take into account details of his particular hardware architecture. As a consequence, software for supercomputers is rarely portable. When the hardware changes, parts of the software need to be re-tuned or even re-written, a process that can take months. Because software is a big investment, users must plan their software development wisely, and decision-makers must protect the user's investment.

Among general software trends, let us comment on the following:

- **Optimizing (tuning and parallelizing):**

Compilers tend to produce efficient code. They are useful particularly if they receive good hints from the programmer, but in general they cannot compete with hand-tuned libraries. In particular for the parallelization, the goal toward a compiler that parallelizes sequential code in a completely automatic fashion, without any manual assistance, has made big progress but we are still far from the dream to completely hide this process to the programmer.

- **Parallel programming languages:**

Although many experimental parallel programming languages have been designed and implemented, these are not generally available to the public, in particular not for a large variety of platforms.

Users are therefore bound to the usage of parallelization paradigms based on standardized parallelization directives as OpenMP, or standardized parallelization libraries as the Message Passing Interface (MPI) supported by most hardware vendor.

- **Program libraries:**

Computer science researchers and hardware vendors have put much effort into the development and tuning of software libraries. Well-known libraries are available on different systems and run efficiently on a wide range of architectures, and their importance will keep growing. The extensive use of program libraries is the most effective way to program



scientific computations. One reason for this is that algorithm experts write library programs, and often an improved algorithm can achieve much larger gains in execution speed than just buying a faster computer.

- **Self tuning software:**

Today's high-performance software, tuned to a particular machine, is typically hardly portable. In future, "self-tuning software" [Cipra, 1999] may partly overcome this handicap. Self-tuning software probes the hardware to determine various machine parameters and then generates code that takes advantage of what it finds. Experiments with well-understood algorithms such as matrix multiplication and Fast Fourier Transform look promising. This idea will likely be extended to many other standard algorithms.

### *Supercomputers in a supercomputing environment*

It is not quite obvious to clearly define what a "supercomputer" is, since lots of different metrics and opinions exist. The "Top 500" list defines a supercomputer on the basis of a single value (i.e. the achieved performance on the famous Linpack benchmark) and it has become a relatively "standard" classification method. However one could argue that one number only is not sufficient to describe and also classify extremely expensive computers. Why not measure the performances achieved on a large number of computing- and scientific- relevant applications? Or why not classify computers according to price/performance, heat consumption/performance or footprint/performance ratio? All these possible classifications are only focused on a "single entity" that might be part of a much more complex system: the supercomputing environment.

It is not trivial at all to evaluate whether, given a certain metric, one of those entities that might be classified as a supercomputer could well fit into a much more complex environment. Typically, supercomputing centres have quite significant requirements in terms of overall reliability, system integration (high speed networks, LAN and WAN) and large data handling (disks and archive systems). Are all supercomputers good candidates for such an environment?

Pure hardware components, it does not matter whether cheap or expensive, are not sufficient to guarantee the success of a supercomputer within a complex environment: the OS is most probably one of the major components. For instance, single image features on large tera-scale systems that allow unique management operations, resource view, efficient job management and scheduling are also highly important, as well as global and parallel file system functionalities.

Not less relevant are also compilers capabilities, which together with the OS can add a quite significant value to a given supercomputer. The efforts put from many vendors in all those fields cannot usually be found together with off-the-shelf made (super)computers. Furthermore all these components are typically supposed to be effective within a "heavy" multi-user/multi-application environment. This consideration is most probably the key issue to be addressed when speaking about supercomputers in general and can be a significant killing factor for all those supercomputers that are - at least on paper - very attractive.

### III.2.3 Conclusion and Recommendations

The main HPC facilities available now and most probably in the next years to come are based on clusters of SMP (most of them microprocessor-based, shared-memory inside nodes, distributed-memory between nodes, complemented with fast node interconnect) built either out of RISC or vector CMOS CPUs. Such platforms might have complex memory-hierarchies, often memory

bandwidth and latency issues dominate performance. For the PRISM system the first priority must be placed on making the PRISM components run efficiently on these platforms.

Targeting portability on several HPC facilities the PRISM system is likely to be run on both vector and RISC-based CPUs. Ideally, the code would have the flexibility to run efficiently on both. In practice, this can be sometimes difficult to achieve since the choice of optimal data structures, loop ordering, and other significant design decisions may differ depending on whether code is intended for vector or RISC CPUs.

### III.3 Software Engineering Process, Coding Rules and Quality Standard

*It is recommended that PRISM should follow industrial software development strategies. Pro's and Con's of this approach are discussed.*

---

The software/system engineering processes should follow industrial specified standards. A basic flow of the engineering process is on a purely technical base:

- System requirement analysis
- System partitioning
- System level requirements for software verification and validation
- Integration of software

Software operations with frequently performed validations

Major aim of this process is to provide functional, stable, and appropriate usable software. A definition and description of the whole software engineering process is specified by the European Cooperation for Space Standardization in ECSS-E-40A (13 April 1999) - Space Engineering Software - published by:

ESA Publications Division  
ESTEC, P.O. Box 299  
NL-2200 AG Noordwijk  
The Netherlands

or accessible via <http://www.ecss.nl/>.

This publication contains all required justification for proper software development.

#### *Design options*

Optimal design of the PRISM system would be that all components are following industrial specified engineering processes and quality assurance schemes.

Another option would be to leave existing components on the current level of development, and apply the proposed engineering process and quality assurance scheme on newly written software only.

The last option would be to follow the scientific community 'business as usual' concept without following engineering process and quality assurance schemes.

## *Constraints*

Several structural problems are limiting the potential of building proper standards following software. These problems are coming on one side from the ignorance of funding agencies on one hand and the basic funding institutions on the other side regarding highly complex software development. Expecting that students, PhD students, or scientist are capable software engineers for such difficult tasks is not appropriate. They are trained and educated for very different tasks. Unfortunately this is still a very common expectation and praxis.

# Architecture Choices, Detailed Design and Implementation

ARCDI



# ARCDI

## I - Basic Choices

*The components of the PRISM system and their general relations are defined. Basic scientific principles and global parameters that need to be consistently defined in all components are discussed.*

---

### I.1.1 PRISM Components

PRISM aims to bring models of subsystems of the Earth system into a common framework. For the time being, the following components will be included in PRISM: atmospheric general circulation models (AGCM), atmospheric chemistry models (AC), ocean general circulation models (OGCM), models of the ocean biogeochemistry (OC), land surface (LS) and sea ice (SI) models. All of these components can be either global or regional models. PRISM is an open, modular system. It is envisaged that additional components can be included at a later stage. This may e.g. include ocean waves, continental ice sheets or models of volcanic or solar activity.

### I.1.2 Conservation Principles

It is essential that the PRISM Earth system model components obey all known conservation laws. When coupling models, it is important that there are no non-physical sources or sinks. All conservation laws have to be satisfied point by point up to the accuracy of the algorithms used. Consistency of fluxes across interfaces is essential, and the exchange of quantities between component models must not only obey the respective conservation laws, but also go along with the change of associated quantities in the source model(s). This also ensures global conservation. These prerequisites are essential to allow the PRISM system to run stably for long time periods.

### I.1.3 Universal parameters

Parameters common to more than one PRISM component need to be consistently defined. These include:

(Please turn)

<b>Astronomical</b>	Earth radius Calendar used (this determines length of day and year and the Earth's angular velocity)
<b>Physical</b>	Gravity acceleration Solar constant and its variability Latent heat of fusion/evaporation for ice/water Density of pure/sea water Specific heat of pure/sea water Reference density of sea water Stefan-Boltzmann constant Full equation of state for sea water (density as a function of temperature, salinity and pressure)
<b>Model parameters</b>	Initial and stop date of experiment/run Length of integration Frequency of saving restart/analysis files

Table 15: Examples of universal parameters

The choice of the model calendar and time control is particularly important. While the proleptic Gregorian calendar (365/366 days) is preferable, it is essential that other options are not excluded in the PRISM specifications. Tools for the conversion of different calendar options need to be defined. The same calendar must be used by each component in a coupled simulation.

#### Further recommendations

It is desirable that one component of PRISM can be coupled to simplified versions of other components (e.g. soil water and energy sub-component, simplified chemistry, mixed-layer ocean or thermodynamic sea ice coupled to the atmosphere). The PRISM system should be able to preserve fast communication with these simplified components.

Sea ice and the upper ocean are to be regarded as a unity, irrespective of whether they are split into modules or contained in the same piece of code.



## II - A proposal for standard Atmosphere/Ocean/Sea ice interfaces

*This document describes a first proposal for standard Atmosphere/Ocean/Sea ice interfaces. The introduction of new independent modules significantly simplifies the exchanges and helps to ensure they are "process-based". Several issues are discussed and comments from the modeler's community are now invited.*

---

### II.1 Introduction

For historical and practical reasons present-day physical interfaces very often are results of an ad-hoc approach. As integrated earth system models are increasingly used for climate studies and prediction, the need for physically based standard interfaces between their components becomes critical. Hence, the research community is faced with both a challenge (to design and agree on standard interfaces) and an opportunity (the awareness of the community and the availability of funding). We here present the result of a yearlong exchange between a number of modelers of the atmosphere, the ocean, the sea-ice and the land surface. The coupling between the first 3 components is so intricate that the corresponding interfaces need to be design together. An atmosphere/land surface interface has already been proposed by the PILPS project (Polcher *et al.* 1998). Other documents will focus on the interfaces with the atmospheric chemistry and the ocean biochemistry. The design of a physical interface design clearly does not have a single optimal solution. Every proposition is a compromise between many physical, numerical and practical constraints. Besides making such a proposition, the present document aims at identifying and establishing a hierarchy of these constraints.

The document is organized as follows: after identifying a number of design constraints, we propose a first version of these interfaces, introduce new modules and raise several issues. An appendix to ARCDI II on page 229 provides the details of all the fields exchanged.

### II.2 Interface design constraints

#### II.2.1 General constraints

Following the PRISM system specifications, the two main scientific principles concerning the present interfaces are:

- Local and physically based conservation of fluxes across coupling interfaces
- Consistent global conservation of energy, water and ocean salt

Accordingly and based on the PILPS experience, we chose the following criterion for standard interface design:

- Identify physically based interfaces across which the conservation of energy, mass and momentum can be ensured.

- Identify which process needs to be computed by which component/module and ensure that there is no duplication or inconsistency in these computations.
- Identify numerical constraints.
- Stability: Neumann vs. Dirichlet boundary conditions, impact of different time steps (components, coupling).
- Interpolation: sub-grid scale heterogeneity issues, local conservation,.. all also depending on the ratio of atmosphere to ocean grid resolution (a working assumption is that ocean and sea-ice share the same grid, see below).
- Identify historical and practical constraints not likely to evolve in the next five years.

## II.2.2 Model components constraints

The following model component constraints are identified. They can be physical, numerical or practical.

### II.2.2.1 Atmosphere

Stand alone integrations of the atmospheric model forced with observed or simulated ocean/sea-ice fields should be possible.

Coupled simulations with a slab ocean/sea-ice on the atmospheric model grid should be possible.

Atmospheric model grid may be defined independently than the ocean/sea-ice models since it may depend on numerical or physical constraints (conformal grid, increased resolution over a specific region...).

For stability of numerical schemes, the treatment of atmospheric turbulence, coupled with the calculation of surface fluxes may be implicit.

The dependence of latent heat with temperature or specific heat with water content should be consistent within the atmosphere and at the surface.

### II.2.2.2 Sea ice

The sea-ice and ocean models grids should be the same. This is important to ensure conservation of heat, salt, freshwater, and momentum locally as well as globally. It is also crucial for appropriately resolving the sea-ice front and the horizontal fluxes of freshwater at the interior of the ice packs.

The sea-ice and ocean model grids currently differ from those of atmospheric models.

All the existing sea-ice models include a parameterization of leads, so that these models need to know the surface fluxes of heat over both sea ice and ocean.

During a time step, the ice concentration is modified by ice dynamics, which is usually called first for both physical and numerical reasons, and then by ice thermodynamics.

In the near future, sea-ice models will include several ice types and/or ice-thickness categories within a grid cell.

### II.2.2.3 Ocean

To ensure both local and global conservation of fluxes, the ocean surface of both ocean and atmosphere grids need to be the same and is provided by the ocean component. This requires a tiling scheme in the atmosphere component.

Volume conservation of the ocean requires that the thickness of the first ocean level be variable and function of the mass of floating sea ice.

The coastline, the bathymetry and the extension of polar caps are considered as fixed during any coupled integration.

## II.3 The Proposal

From these constraints, we propose the interfaces described in Figure 16. One main difference with present day interfaces is the introduction of new modules (detailed below). This added modularity simplifies the exchanges, ensures they are “process-based” and helps distinguish fast and

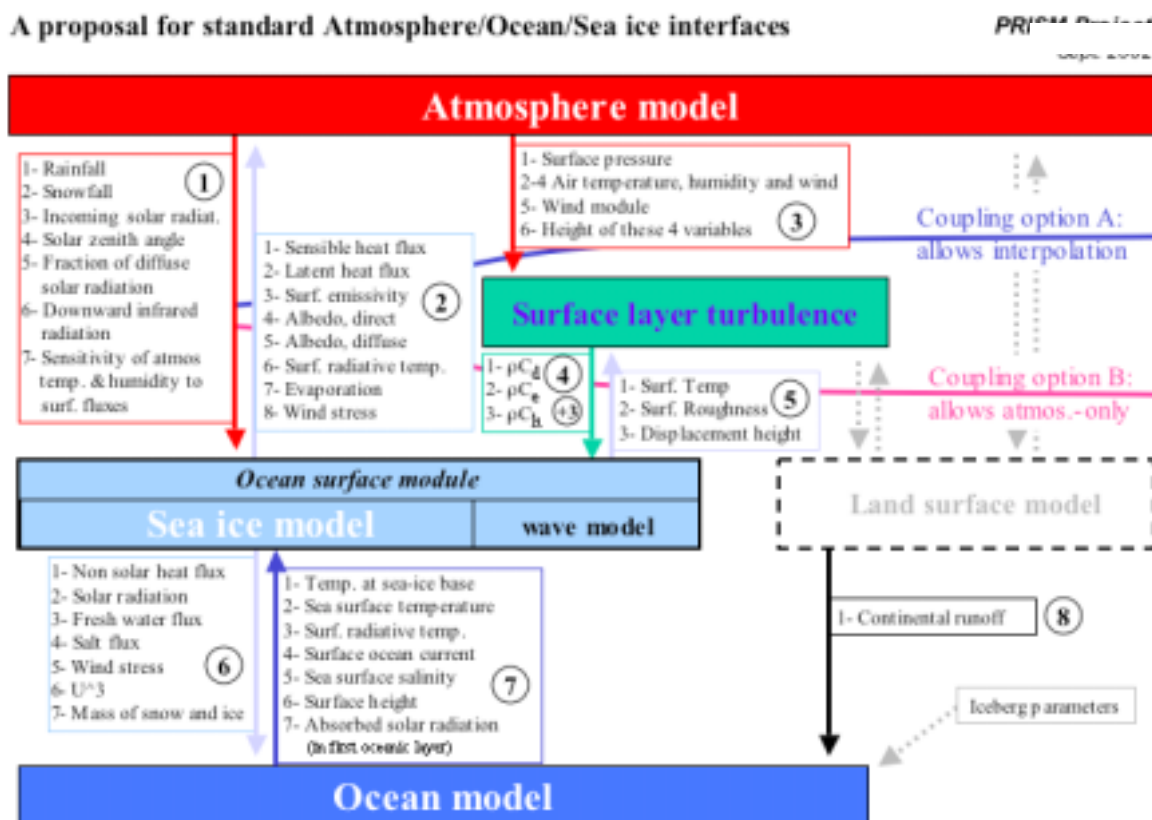


Figure 16: A proposal for standard interfaces

slow processes. The only exchanges considered here are represented by groups of fields attached to solid arrows in colour and numbered from **1** to **8**. Grey dotted arrows suggest how other components (like the land surface schemes) could fit in. In the remainder of the text each field is identified with a code it I.J where *I* is the exchange number and *J* the field number. For example field 2.2 is the evaporation and field 7.4 is the surface ocean current. The Appendix on page 229 provides the complete list of these fields, including their precise definition and units.

## II.3.1 Introducing new modules

The need for new modules came early in the discussion. Indeed they allow to 1) clearly identify, where the computation of some physical processes happens and 2) easily control unstable computations by distinguishing where fast and slow processes are computed. Increasing the modularity is a long-term goal for most components but we restricted ourselves to two key modules for practical reasons.

### II.3.1.1 The Surface Layer Turbulence module

The Surface Layer Turbulence module (SLT module) contains the description of the turbulence inducing diffusion in the surface layer of the atmosphere (above the ocean + sea ice system). It computes the surface layer turbulent coefficients (4.1, 4.2 and 4.3) from the surface boundary conditions (exchange **5** provided by the ocean-surface module, see below) and the atmosphere prognostic variables at lowest level (exchange **3**). It also provides the atmospheric variables of exchange **3** to the ocean-surface module, which needs them for some computations (see next section). As it is non-physical to interpolate the turbulent exchange coefficients, they need to be computed on the finer grid (ocean + sea-ice). As atmosphere models also need this module for classic stand alone forced integrations, we propose two coupling options (Figure 16): **A**, which avoids interpolation of turbulent coefficients and **B**, which allows atmosphere-only simulations, ensuring the historical coherence with previous integrations. Note that:

- The bulk formulas used to compute the surface turbulence might differ in the two options.
- The SLT module has to produce the atmospheric boundary layer diagnostics (fields at 2m for instance).

### II.3.1.2 The Ocean-Surface module

This module is introduced to help separate the fast ocean+sea ice surface processes, involving heat, water and momentum exchanges with the atmosphere and the sea ice from the slower, deeper processes. It acts as a homogeneous filter between the atmosphere (directly or via the surface layer turbulence module) and the ocean+sea ice system. It computes a number of surface fields (wind stress, sensible heat flux...) using bulk formulas. It receives fields from and produces field to: the atmosphere (exchanges **1** and **2**), the surface layer turbulence module (exchanges **4** + **3** and **5**) and the ocean (exchanges **6** and **7**). In a first stage, this module will in practice be the sea-ice model (with of course no modifications of fluxes over open ocean). In a second stage, a wave model will be included to provide sea surface roughness (field 5.2) to the SLT module.

## II.3.2 Physical interfaces

### II.3.2.1 Energy fluxes computations

*Between ocean-surface module and ocean component:*

The ocean-surface module, via the sea-ice model, provides the ocean model with the net solar radiation (6.2) and non-solar heat flux (6.1) entering the ocean surface. In return, the ocean model provides the ocean-surface module with a) the temperature at the sea-ice base, b) the sea-surface temperature, c) the sea-surface radiative temperature, and d) the fraction of solar radiation absorbed with the first oceanic layer. The sea-ice model uses first of these fields (7.1) to compute the oceanic heat flux at the ice-ocean interface. Note that in current models (even in those that account for the depression of ice below the water level), this temperature is taken as the sea-surface temperature. The second (7.2) and third (7.3) fields are required for the calculation of the atmospheric turbulent heat fluxes and long-wave radiation over leads, respectively. Finally, the last field (7.7) is needed for the computation of the energy budget of leads by the sea-ice model.

*Between ocean-surface module, SLT module and atmosphere component:*

The atmosphere component provides the ocean surface module with the incoming solar radiation (1.3), possibly for different spectral intervals, the solar zenith angle (1.4) (or its cosine), the fraction of diffuse solar radiation (1.5) and the downward infrared radiation (1.6), for a complete calculation of the radiation budget either over free ocean or over sea-ice. It also provides the ocean surface module with the sensitivity of atmosphere temperature and humidity to surface fluxes (1.7) to allow an implicit calculation of surface fluxes. In order to evaluate the exchange coefficients and then the turbulent fluxes, the atmosphere component provides the surface layer turbulence module and the ocean surface module with surface pressure (3.1), air temperature (3.2), specific humidity (3.3), wind components (3.4) and mean scalar wind speed (3.5), and the height of the level where all these parameters are calculated (3.6). The mean scalar wind speed will possibly include gustiness effects due to free convection in boundary layer or due to deep convection. The ocean surface module provides the atmosphere model with the turbulent energy (sensible and latent) fluxes (2.1 and 2.2). The emissivity (2.3), the albedo for direct and diffuse radiation (2.4 and 2.5), possibly for different spectral intervals consistently with the partitioning of incoming solar radiation, and the surface radiative temperature (2.6), all calculated by the ocean surface module to solve the surface radiation budget, are also transferred to the atmosphere component. The surface radiative temperature may be different from the surface temperature communicated by the ocean surface module to the surface layer turbulence module for the calculation of exchange coefficients. This last temperature, for free ocean areas, may be indeed representative of the upper mixed layer (depth of a few meters) as this is the case when evaluating and calibrating the parameterizations of the exchange coefficients.

### II.3.2.2 Mass fluxes computations

The atmospheric model provides the ocean-surface module with rainfall (1.1) and snowfall (1.2), as the sea-ice model needs both fields to run. In return, the ocean-surface module provides the atmospheric model with the evaporation/sublimation averaged over the oceanic grid cell (2.7). This field is needed for the calculation of the hydrological cycle in the atmosphere, since it cannot always be inferred from the latent heat flux (2.2) (due to a possible combination of ocean and ice within one grid mesh, and due to a possible dependence of the specific latent heat with temperature). The ocean-surface module transfers to the ocean model the net fresh water flux (6.3), the net salt flux (6.4) and the total mass of snow and ice (6.7). The net freshwater flux (6.3) results, on one

hand, from the net atmospheric water flux over open ocean (Rainfall+Snowfall-Evaporation) and, on the other hand, from snow melting on top of sea ice, ice growth/melting, snow-ice formation, runoff of rainfall through sea ice into the ocean, snowfall and rainfall over leads, and evaporation over leads. The net salt flux (6.4) is provided by the sea-ice component and results from ice growth/melting and snow-ice formation. The total mass of snow and ice (6.7) is provided to the ocean model to compute the depression of ice below the water level. In return, the oceanic model provides the ocean-surface module with the sea-surface salinity (7.5), which is used by the sea-ice model to compute the freezing point of seawater and, in the future, the salinity of newly formed sea ice and snow ice. The land-surface model provides the continental run-off to the ocean (8.1).

### II.3.2.3 Momentum fluxes computations

The ocean-surface module computes the surface turbulent wind stress. The computation uses the turbulent exchange coefficient (4.1) provided by the SLT module, the wind at the lowest level (3.4), its module (3.5) and its height (3.6) provided by the atmosphere via the SLT module, and the surface ocean currents (7.4) provided by the ocean. The ocean-surface module then transfers the wind stress to the atmosphere (2.8) and to the ocean (6.5). The ocean-surface module also computes the “wind work”  $U^3$  (6.6) and provides it to the ocean.

## II.3.3 Discussion

### II.3.3.1 Scientific issues

Standard vs. coherent: some models do not provide all the necessary fields or provide them at different levels (2m vs. lowest level). The SLT module solves this problem for the computation of turbulent fluxes. Accordingly, it requires the height of the lowest level (field 3.6).

Frequency of exchanges: each process needs to be computed at the relevant physical time scale. This requires isolating some processes so that each GCM can run at its optimal time step. Namely the frequency of exchanges **1**, **2**, **3**, **4** and **5** can follow the atmosphere time step while exchanges **6** and **7** only need to be done at ocean time step frequency (which should be a multiple of the atmosphere frequency).

The temperature of the fresh water flux (E-P+R) should (and can) be taken into account in the ocean. Can the atmosphere and land surface components provide it while ensuring a coherent energy balance?

The formulation of latent heat as a function of temperature or thermal capacity as a function of water contents should be consistent in all components and modules.

### II.3.3.2 Practical, technical and other issues

Adaptation of models towards a standard interface: a scientific priority for all groups?

Computer cost issues: in practice the implementation of such interfaces most likely involves a trade off between physically based processes and fast computation. Many of today's ad-hoc interfaces are efficient and moving to the interface we propose here would probably degrade the performances. Nevertheless, 1) we assume a net increase of power in the year to come and 2) the design forward-looking earth system models should only rely on sound physical and numerical bases to avoid jeopardizing future developments.

### III - System Architecture

*This document is primarily about the infrastructure necessary to enable a distributed configuration, submission and monitoring system for PRISM experiments. A local system will be delivered to enable model developers to run the configuration tool without the web service infrastructure to test out their models. The local system is based on the same software to ensure a transparent transition between the local and the Web Services infrastructure system. The requirements and constraints from the REDOC document are translated into a specific design in the document. The architecture is detailed into software components and the choice of these components is discussed. A run-time and deployment architecture is discussed and finally an implementation model detailing the order and importance of implementation tasks is presented.*

---

#### III.1 Introduction

The purpose of the PRISM system is to enable users to perform numerical experiments, coupling interchangeable model components using standardized interfaces.

The general architecture provides the infrastructure for this. There is an emphasis on choosing an architectural design that allows these activities to be done remotely, e.g. without the user physically being in the place where the numerical computations take place.

Within the PRISM system three different types of actors exist:

**Users:** someone who uses the PRISM System to assemble and run a coupled model.

**Model developer:** someone who makes changes to a PRISM model code that will be introduced to the default model version.

#### Proposed PRISM architecture (generalized web services)



Figure 17: PRISM user interacting with the central and local PRISM sites in a Web based scenario

**Administrator:** the person who provides the link between developers and users.

These different actors represent different demands on the system. The best design in order to fulfil the PRISM requirements (remote access, modularity, extensibility) is a centralized architecture that minimizes the administration and duplication of resources at the expense of some added complexity (see Figure 17).

It is proposed to gradually refine the system in two phases and into two different products configured from the same software base:

**Local system:** used by developers to develop models without affecting operations.

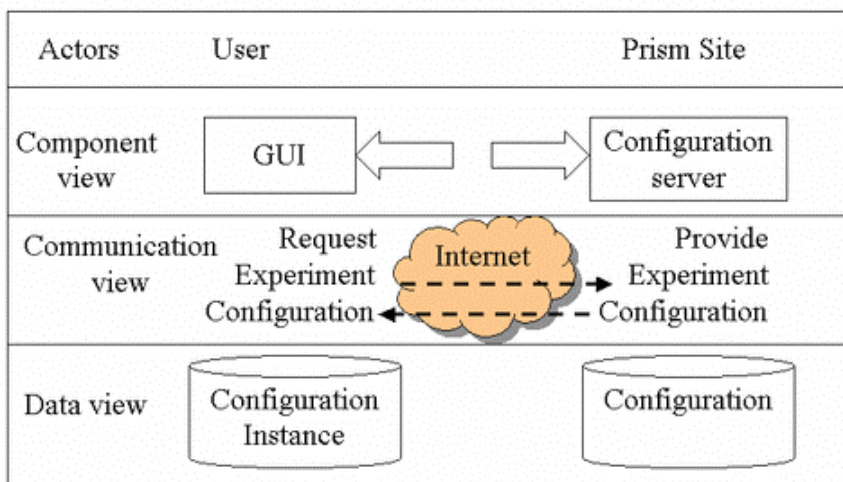
**Web based system:** used by administrators and users to administrate model changes and run coupled experiments remotely.

### III.2 Terminology and Concepts

**Software component:** software that is a part of a larger system, or an object that is part of a larger object, delivering a specific functionality.

**Deployment:** describes the physical distribution of the system being built, in terms of how functionality, i.e. software components, is distributed among a set of run-time processing nodes (computers). The term is also used in a more limited scope in other documents referring to the deployment of an experiment to computing resources (REDOC II.2).

## Configuration process



**Configuration:** The arrangement of a computer system or component as defined by the number, nature, and interconnections of its constituent parts. Also used in relation to experiments where it defines the collection of programs, documents and data that an experiment consists of. Figure 18 shows different aspects of the configuration process between the PRISM user and the PRISM site.

Figure 18: Configuration process in PRISM: Different views of the configuration process that are detailed in the document relating to the User and the Prism site.



### III.3 The Local System, a Basis for Development

A large amount of effort in the PRISM project will go into modularization of models and standardization of scientific input and output. Likewise, the task of deployment of models on the web infrastructure and also the task of making it a painless exercise for a user or developer to download a model locally and make it work needs to be given the same attention and priority.

To succeed in the deployment of models we have to standardize procedures, i.e. the standardization of model builds, model runs and model configuration is critical in view of long term maintenance and usability of the system.

A file structure must be specified that enables a locally configured model to be put into an archive file and transported to a deployment site and deployed there automatically.

Such a structure needs to be proposed by the model developers but could look like this:

```
/prism
  /models/modelX/src
    /lib
    /doc
    /config
    /runscript
    /bin
    /tmp
```

A version of the GUI (Graphical User Interface) that runs without the web infrastructure will be developed initially. This version can be started from the command line on the users linux workstation and will not need any network support. Support for other platforms is optional. All input and output will be read/written to the current directory according to the standard structure described above. The user will have to download the latest versions from a central source code repository.

The purpose of the local version is:

- To allow individual developers to adapt, test and run their models locally.
- To enable testing of build and deployment scripts locally.
- To prepare a version for deployment in the web infrastructure.

Once a model works properly it can be made available for users through the Web Services infrastructure, which is described in the following section.

### III.4 Architecture of the Generalized Web Services Concept

A Web Services (WS) system makes the services it provides abstract, shielding the client from the particularities of the implementation. A client, be it another service or a user interface, requests a specific service. This is located with the help of a WS registry using UDDI, the Universal Descrip-

tion, Discovery and Integration protocol, which contains information on where the service is located and how to use it as shown in Figure 19.

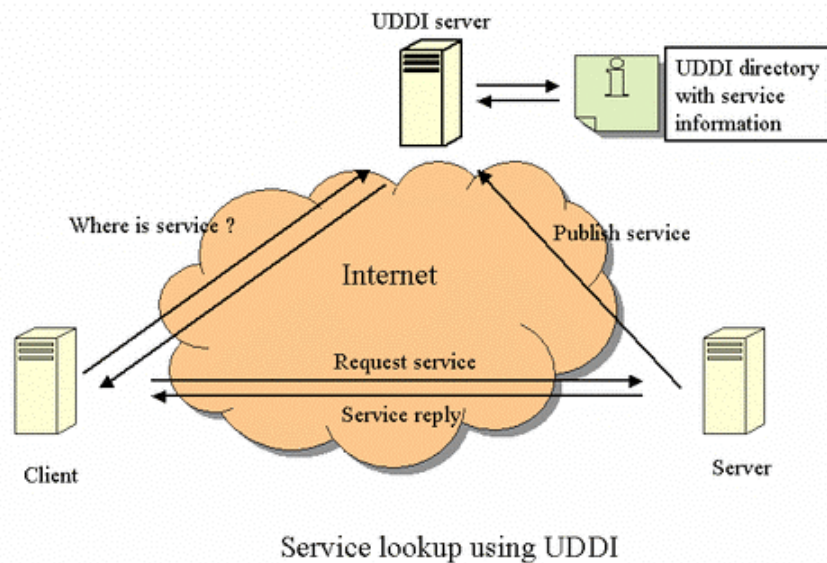


Figure 19: Service lookup in WS: Client request of a service from a server by looking up the service provider in a directory.

A WS provider uses the UDDI protocol to publish the service it provides encoded in WSDL, the Web Services Description Language to a UDDI directory.

The client requests a service from that directory. The directory information returned tells the client how to find and use the service. It is the intention to use this system to allow users to find local PRISM sites.

### *Subsystem topology*

A wide area network (WAN) is a geographically dispersed telecommunications network. The term distinguishes a broader telecommunication structure from a local area network (LAN). A WAN may be privately owned or rented, but the term usually indicates the inclusion of public (shared user) networks.

A LAN is a group of computers and associated devices that share a common communications line and typically share the resources of a single processor or server within a small geographic area (for example, within an office building). A LAN may serve as few as two or three users (for example, in a home network) or many as thousands of users.

The client is typically running in his LAN requesting a service from applications running in another LAN using a WAN such as the Internet and the related protocols to forward the request.

The WS universe consists of a number of zones or locations derived from its networking status as LAN or WAN .

The importance of the zones lies in that as messages travel from a LAN to a WAN the authentication and validity of the message is lost because the public network potentially allows tampering with the message. Measures need to be taken to detect and prevent such tampering.

### Subsystem layers in WS

Given here are the software components in the architecture of the subsystems infrastructure for a generalized WS system. The application layer describes the software component(s) in the subsystem and its function. The middle-ware layer details the actual implementation of the subsystem as shown in Figure 20.

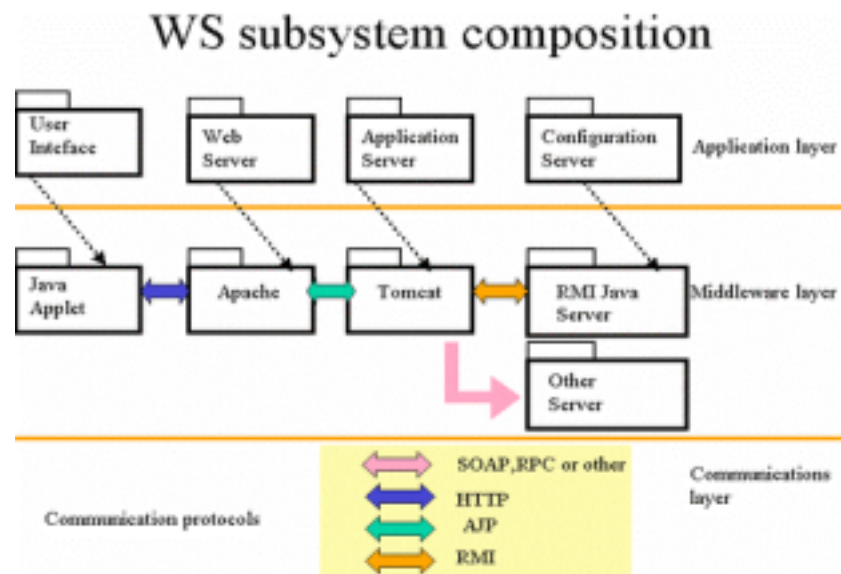


Figure 20: WS subsystem composition: Layers that make up the subsystems in a WS architecture.

Table 16: Software components in the system.

Application Layer	Application Function	Middle-ware Layer	Communi-cations Protocol
User Interface	Interface task	Java Applet	HTTP
Web server	Serves pages, Listens for requests	Apache	HTTP
Application server	Interface application	Tomcat	AJP12
Domain Applications	Deliver domain function such as configurations	Any	Any
Authentication server	Authenticate clients	S/key, Kerberos	HTTP/HTTP S
ORB (Object Request Broker)	interface to application server	Java	HTTP
Event service	Deliver events to applications	Java	HTTP
Directory server	Publish and access information on services	UDDI server Java	WSDL
PRISM administrator	Executes administration tasks	Definitions of administration entities	SSH

The implementation choice is decided on deployment of the system, if the particular implementation does not restrict the functionality of the subsystem. A discussion of the choices listed can be found in the deployment section.

### III.5 From Analysis to Design: The Process-to-Component-Translation of the PRISM System

The processes described in the REDOC II - document on page 43 will be translated into software that enables the desired activities.

#### *Process-to-Component View*

The processes listed in REDOC, Table 12: Table of different processes (1), (2) and (3), referred to as RDT1, RDT2 and RDT3, describe what activities take place and which process is handling it. For each process the software components that handle the related activity is described.

The physical location of the software component also determines the choice of component, but the mapping of components to the architecture will be more closely examined in the deployment section .

The design of the PRISM system processes falls into three categories mirrored by the Tables RDT1/2/3 as shown in Figure 21

**Controller processes:** User (client) activities from Table RDT1

**Controlling processes:** Provider (server) from Table RDT2

**Controlled processes:** Execution of model from Table RDT3

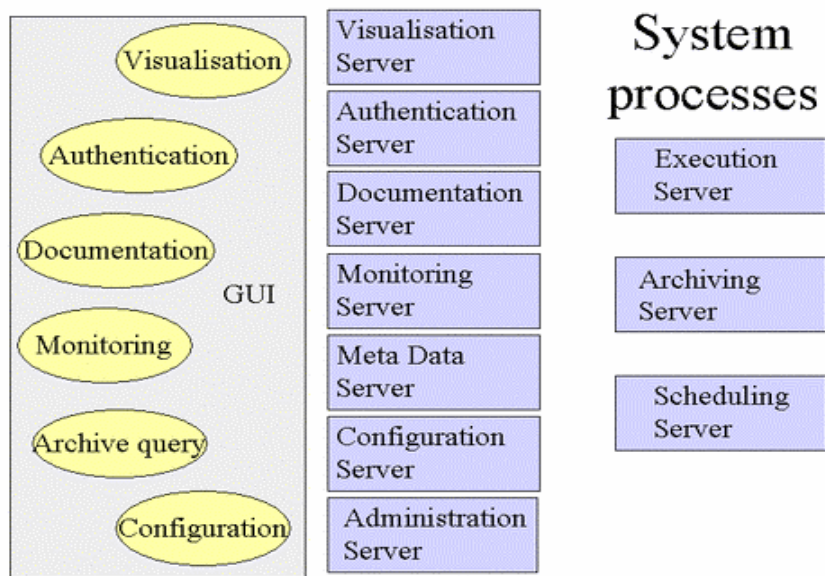


Figure 21 :System processes in PRISM: System processes categorized into controller, controlling and controlled

The Table RDT1 processes naturally fall into the application layer of the GUI in this document as shown in Figure 21.

The Table RDT2 processes are domain service applications that are delivered by application servers directly to the client GUI, Figure 21, middle boxes.

The Table RDT3 processes are the domain applications controlled by the Table RDT2 processes and not served by any application server and, as such, not part of the WS infrastructure, in Figure 21.

### *Subsystem communication*

The subsystems communicate by sending messages to each other. The messages are usually of the form request - reply, but can also be in the form of publishing information, e.g. event notifications. In this case no reply is expected. The term reply should be understood as receiving a useful object back, not a protocol confirmation of the transmission status.

Figure 25 details the messages and protocols in a WS system. The MX numbers in Figure 25 relate to the description of the messages in Table 17.

Message	Protocol	Function
M1	HTTP/S	Load GUI from central web server
M2	HTTP/S	Authenticate from GUI to web server
M3	AJP12/13	All traffic to tomcat
M4	RMI	Reading/writing experiment configurations, submit
M5	FORK	Send job to SMS scheduler
M6	QSUB	Queue job to supercomputer
M7	FCGI	Monitoring requests to SMS on job status
M8	RPC	Cgi server requesting information from SMS
M9	AJP/RMI	M3/M7 traffic if PRISM site has no WEB SERVER
M10	RMI, SOAP, RPC	M4 traffic if PRISM site has no WEB SERVER OR TOMCAT
M11	FTP	Ftp of new model code to build. Requested by Prism site

Table 17: PRISM system messages

Detailed interaction analysis for the following cases, i.e. typical scenarios is made in form of a diagram and a table detailing the resources needed for every message:

- CASE 1: User downloads GUI and directory information. Messages M1, 2, 3, 4
- CASE 2: User submits build job. Messages M1, 2, 3, 4, 5, 6
- CASE 3: Monitoring of job. Messages M1, M2, M3, M7, M8

The three cases are illustrated on the next pages.

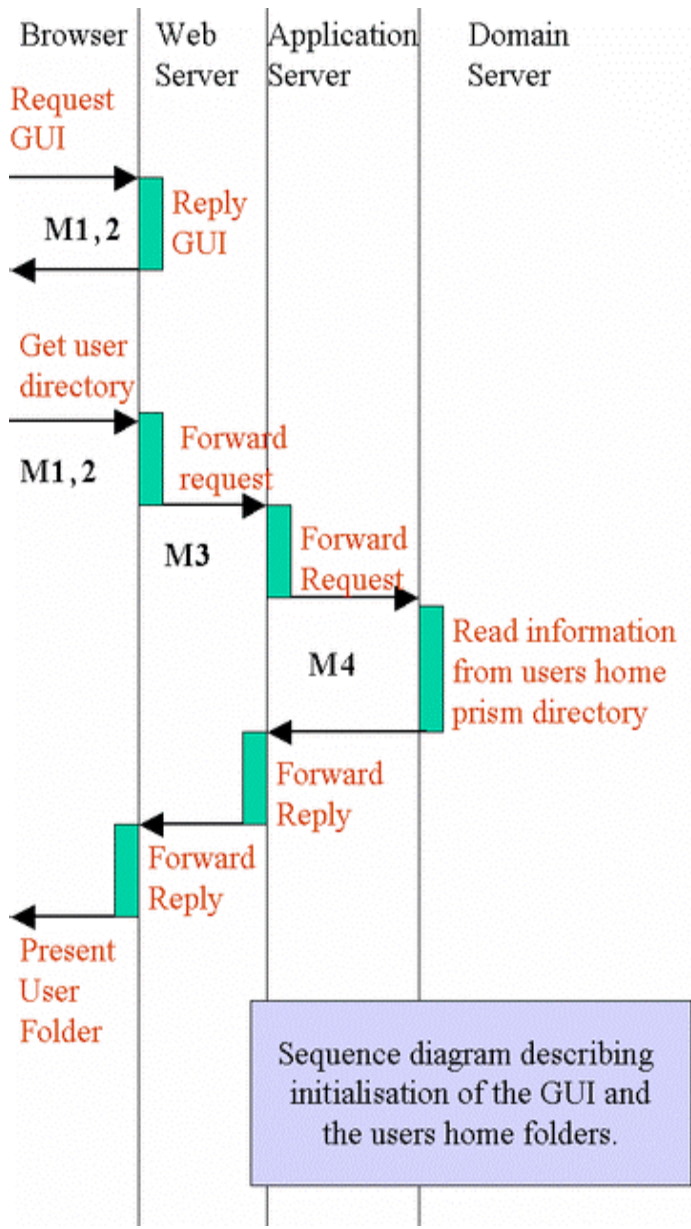


Figure 22: Case 1, User downloads GUI and directory info: Swimlane diagram of the initialization of the users home folders showing the transition between software components.

Message	Function	Files or information needed
M1	Load GUI from central web server	Java classes
M2	Authenticate from GUI to web server	Password information
M3	Request application	Directory information on applications available
M4	Reading users home directory	Directory listing from file system

Table 18: Case 1, this highlights the administration and maintenance advantages of having all users running the same software versions of a program and that all configuration information for compositions is coming from the same domain server and is saved there

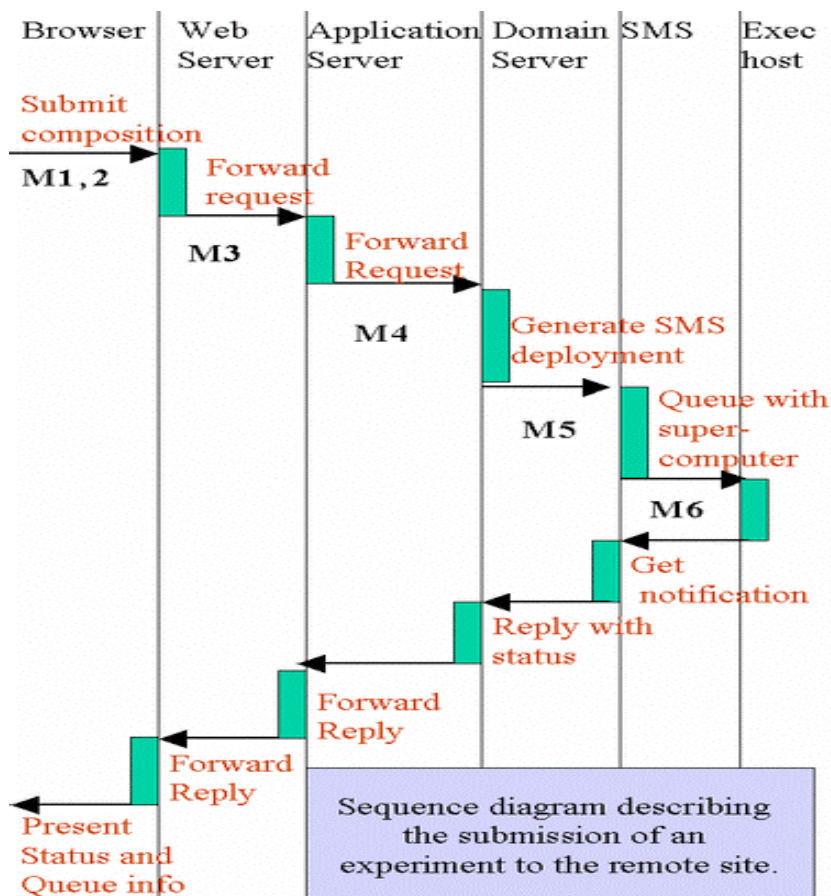


Figure 23: Case 2: User submits experiment; Swimlane diagram of the submission of an experiment to the remote site showing the transition between software components.

Message	Function	Files or information needed
M1,2	Submit composition to central web server	Values of all changed variables
M3	Request application server on specific site	Directory information on applic's on all PRISM sites available
M4	Generate parameterised standard deployment scripts and SMS control scripts.	Standard scripts and all variables in composition.
M5	Submit standard deployment scripts and SMS control scripts to SMS	Standard scripts and all variables in composition.
M6	Deliver standard deployment scripts to supercomputer queue	Standard scripts and all variables in composition.

Table 19 Case 2, this table highlights the configurability of the system where the M3 step could lead to a submission of the experiment on different sites depending on the availability of computing hosts in the service directory (UDDI).

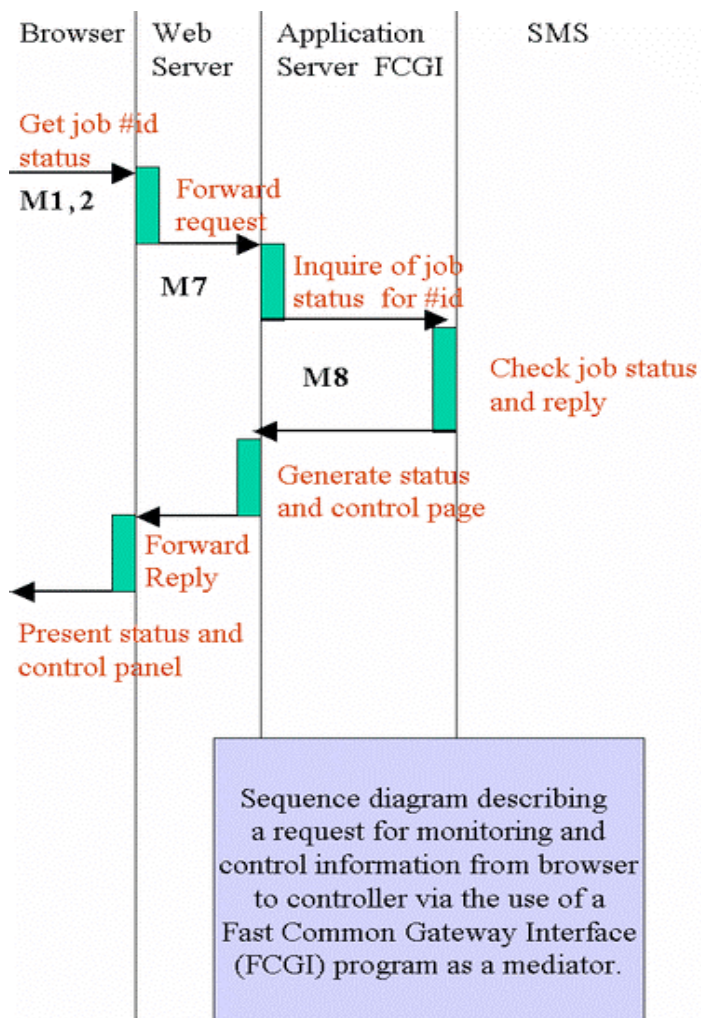


Figure 24: Case 3, User monitors experiment - Swimlane diagram of a request for monitoring and control information showing the transition between software components.

Message	Function	Files or information needed
M1, M2	Request job information from central web server	Job Id
M7	Direct request to application	Directory information on applications available
M8	Request information about jobs	Job and user id

Table 20: Case 3, monitoring and control of an experiment through the use of a mediator program running at the execution host. The role of the program is to make sms functionality available remotely through a web browser.



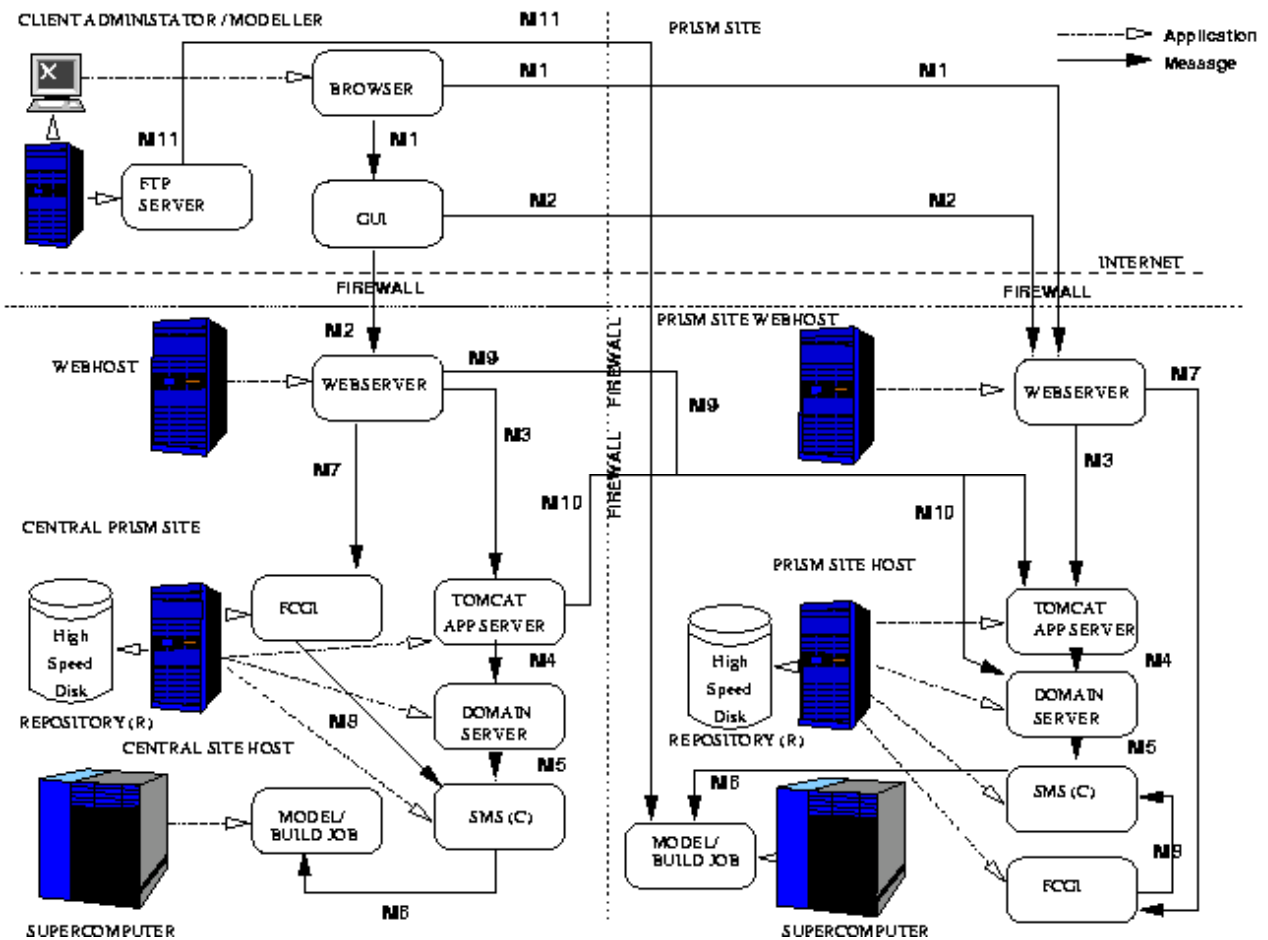


Figure 25: A central site and a PRISM site with all components in place. The system as shown above will be configurable to allow for some components not to be duplicated, see Figure 26 on page 123 and Table 22 on page 128, which lists the placement of the software components in the system. Each process has now been classified into its subsystem software component and the communication between these components has been described.

## Overview of the Software Components

The use of existing components is necessary to achieve the system as described in REDOC in the time frame of the project.

Id	Application	Application layer	Implementation	State
1	Experiment configuration	User interface	PrepIFS	Exists/to be refined
2	Experiment visualization	??	??	As described by REDOC II.3
3	Authentication	**	??	To be developed
4	Archive query	??	??	As described by REDOC II.3
5	Documentation	User interface	PrepIFS	Exists/To be refined
6	Experiment monitoring	User interface	Xcdp	Exists/To be refined
7	Experiment configuration server	Application	Java RMI	Exists/To be refined
8	Documentation server	Web Server	Apache	Exists/To be refined
9	Administration server	Application	Java	to be developed
10	Archive meta-data browser and server	??	??	As described by REDOC II.3
11	Visualization server	Application	??	As described b REDOC II.3
12	Monitoring server	Application	SMS	To be refined/developed
13	Scheduling server	Application	SMS	Existing
14	Execution server	Application	Coupler, Model	Existing, to be refined/developed
15	Archiving server	Application	??	As described by REDOC II.3
16	Authentication server	Application	**	To be decided
17	ORB	Application	PrepIFS	To be refined
18	Request Listener	Application	Java	To be developed
19	Event service	Application	Java Jini	To be developed
20	Directory server	Application	Java UDDI	To be developed
21	Experiment configuration	Definitions Input data	PrepIFS	To be developed
22	Administration scripts	Application	Any	To be developed
23	Semiautomatic deployment of java components	Application	Java	To be developed

Table 21: Software components and implementations. Legend: [Exists] Can be without modification. [To be refined] : The component exists with the correct functionality but will need some modification. [To be developed] : The component does not exist. [\*\*] : To be decided. See Security implementation . [??] : Has yet to be published.

## Choice of Existing Components: Experiment Configuration

PrepIFS as an experiment configuration application consists of several components: GUI, ORB and configuration server. The GUI allows access to experiment configurations, SMIOC and SCC as described in ARCDI IV.1 on page 135, stored on the configuration server, and also contains logic to validate the configuration. Experiment configurations are sent to the configuration server, which generates the correct job control scripts for the experiments and deploys these through SMS, which schedules the experiment for execution.

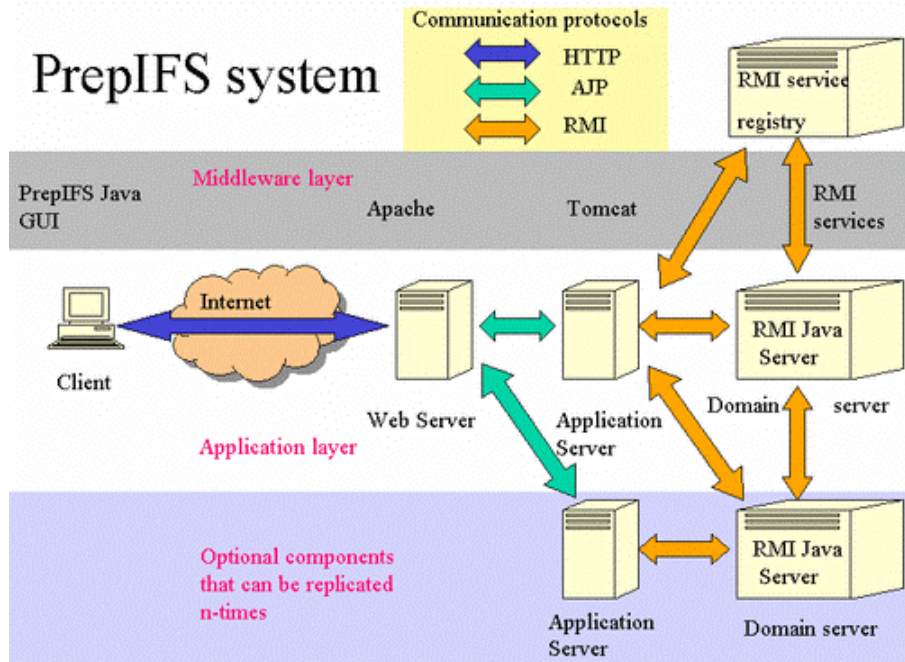


Figure 26: The PrepIFS system and its software components.

The system is built on Web Services and so fits well into the proposed architecture.

## Choice of Existing Components: WEB Server

The choice of Apache is based on the fact that it is free and used by a majority of the world's web servers as well as being reliable, mature, maintained, and having high performance.

## Choice of Existing Components: Application Server

The Tomcat application server is closely linked to the Apache Web server and follows the choice of Apache and Java as implementation language of preference (see REDOC II.1 on page 43) For sites that do not provide a web server or application server, the overhead to install these components is too large. Therefore a service component should be developed that does not require the web server and application server. This service component (we refer to it as a Request Listener (RL)) listens for HTTP requests from the Internet and converts them into calls to the provider ser-

vices listed in Table RDT2. Great care needs to be taken to prove that the RL can only forward authenticated requests and that only the intended services can be reached.

### *Choice of Existing Components: Monitoring and Scheduling*

The choice of SMS/Xcdp as scheduling/monitoring tool is based on our evaluation of its reliability, functionality, performance and suitability for working with very large numbers of tasks and their dependencies. Figure 27 shows how information is exchanged between the components.

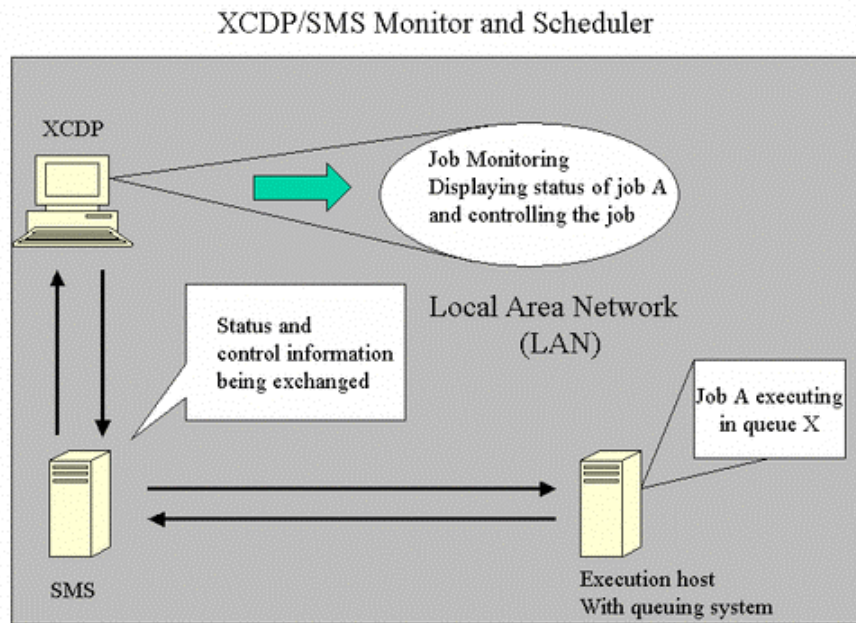


Figure 27: Monitoring and scheduling of experiments; the figure shows how the user can monitor experiments using the Xcdp and SMS tools.

## III.6 Proposed Architecture

Combining the PrepIFS system with the SMS/Xcdp tools results in a configurable and flexible WS system (Figure 21) with much of the required functionality in place as described in REDOC II.1 on page 43.

### *Additions and Changes to Components in Table 21*

The requirements for components listed in Table 21 to be refined or to be developed are examined further.

#### **Experiment configuration , task id# 1:**

Addition of keyboard navigation.

Creation of an experiment configuration definition interface, which allows modellers the remote development of model configurations (optional).

Changes to allow a more flexible experiment hierarchy. Standardization of interfaces.

**Experiment visualization, task id# 2:**

It is assumed that this task will only involve implementation of an interface to start the visualization tools. If there are no requirements the user interface will spawn a browser with a configurable URL.

**Authentication, task id# 3:**

This task contains the development of an authentication system as well as exporting the authentication as an interface to other services if needed.

**Archive query, task id# 4:**

It is assumed that this task will only involve implementation of an interface to start the archive tools. If there are no requirements the user interface will spawn a browser with a configurable URL.

**Documentation, task id# 5:**

PrepIFS has an integrated help/documentation system available. It is desirable that this is extended to allow modellers to add/change documentation remotely in a safe way.

**Experiment monitoring, task id# 6:**

The current monitoring system (see Figure 1) is designed for LAN access and is to be extended to WAN access over the Internet. It is desirable that it retains its graphical view if possible. It should provide the functionality as described in chapter II.4, if this can be achieved without security concerns.

**Experiment configuration server, task id# 7:**

Creation of a deployment environment and the scripts needed to deploy the composed model automatically.

**Documentation server, task id# 8:**

Automatic generation of documentation from all available meta-data and addition of developer descriptions.

**Administration server, task id# 9:**

An interface for administration of the WS system and domain software should be developed. Initially this will be handled manually but some automatic system is desirable. This is further developed in section Software life cycle management.

**Archive meta-data browser and server, task id# 10:**

It is assumed that this task will only involve implementation of an interface to start the archive tools. If there are no requirements the user interface will spawn a browser with a configurable URL.

**Visualization server, task id# 11:**

It is assumed that this task will only involve implementation of an interface to start the archive tools. If there are no requirements the user interface will spawn a browser with a configurable URL.

**Monitoring server, task id# 12:**

Some refinements needed to allow for web based monitoring.

**Scheduling server, task id# 13:**

No changes anticipated.

**Execution server, task id# 14:**

This is the task of the modellers.

**Archiving server, task id# 15:**

It is assumed that this task will only involve implementation of an interface to start the archive tools. Due to the lack of a central archiving system archiving will be made at the site where the experiment runs using the local facilities available. This is detailed in REDOC II.3. If there are no requirements the user interface will spawn a browser with a configurable URL.

**Authentication server, task id# 16:**

This will be implemented as recommended in collaboration with Fujitsu. It should allow for Message Authentication (MAC) between services and for client authentication between users and services.

**ORB, task id# 17:**

The existing ORB has a protocol manager for Java RMI services. This will be extended to allow other protocols such as Java RPC. A typical system is shown in

Figure 28.:

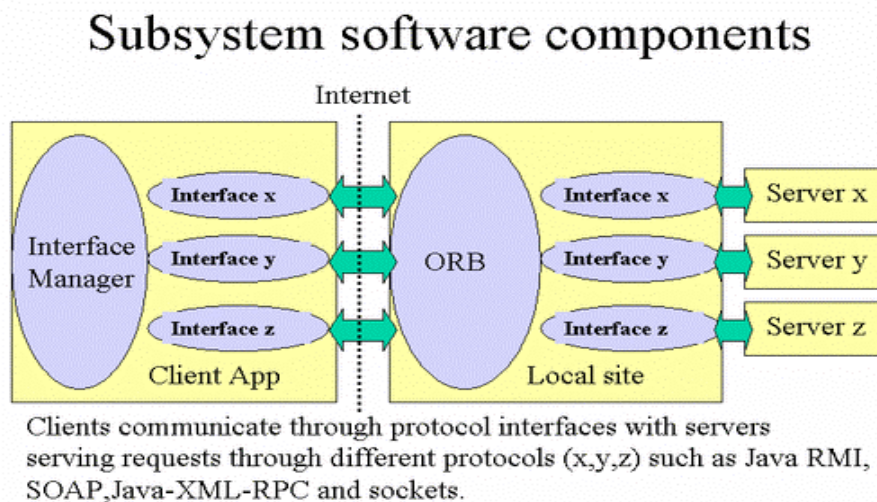


Figure 28: PRISM subsystem software components for communication Software components necessary to interface a client with a server using a specific protocol

**Request Listener, task id# 18:**

This service component, the Request Listener (RL), listens for HTTP requests from the Internet and converts them into calls to the provider services. It allows for deployment of models from the central site to the PRISM sites. There are considerable security implications for this component and by delaying the development of this component to a later stage software to help with this task should have become available.

**Event service, task id# 19:**

An event service should be developed to allow for notification between applications and to clients. A Java application programming interface JINI exists and this will be implemented.

**Directory server, task id# 20:**

The PrepIFS directory server, the interface repository, is a proprietary technology and this should be migrated to a UDDI compliant server.

**Administration scripts, task id# 21:**

The administration model proposed requires considerable standardization and automation. The process is described in section Software life cycle management.

**Experiment configuration, task id#22:**

The use of XML in definitions of meta-data requires the configuration server to be able to read its input data in this format. It might be needed to create special configurations files for this purpose.

**Semiautomatic deployment of Java components, task id# 23:**

An interface for administration of the WS system and domain software should be developed. Initially this will be handled manually but some automatic system is desirable. This is further developed in section Software life cycle management.

*Setup of PRISM Central Site*

The configuration of all system components on a central site on dedicated hardware and with security measures in place.

### III.7 Specifications, Definitions and Standardized Interfaces Needed for the Implementation of the System

1. Format of experiment configurations (definitions and compositions) for instances and inherited values.

2. Format of experiment names and id's for archived data to facilitate unique names.
3. API for experiment configuration widgets in the GUI to allow new visual representations.
4. API for getting context and authentication information from the GUI.
5. API for service extensions and additions.
6. Format of naming conventions used to specify services.
7. Format for calling services from clients, message standard.
8. Development of standards for achieving remote build capacity for each model component. This includes build scripts capable of determining compiler versions and switches, libraries and operating systems.

### III.8 From Design to Implementation: The Component-to-Deployment Translation

The web services concept is a very flexible architecture and well suited for deployment over multiple computers. The proposed architecture involves the partitioning of functionality into a central system for storing experiment configurations and local PRISM sites for executing the models. The components will be located as indicated in Table 22.

Component	Central site	PRISM site
Experiment configuration server	1	0
Documentation server	1	0
Administration server	1	0
Meta-data server	1	0
Visualization server	0	1
Monitoring server	0	1
Scheduling server	0	1
Execution server	0	1
Archiving server	0	1
Authentication server	1	0
ORB	1	0
Event service	1	1
Directory server	1	0

Table 22: Software components and their location

Some replication of functionality is possible, see Figure 26. It is desirable to store the experiment configurations centrally to minimize administration unless the system is run locally.



## *Detecting, handling and reporting faults strategy*

Large parts of the communication will take place over public networks (Internet) of which we have no control. It must be understood that communications over the Internet is on a best effort basis. An organization (emails etc.) for reporting faults in web services together with problem report standards must be set up. A monitoring system for the web services together with some diagnostic tools would be desirable.

## *Integration with other Work Packages, Published Interfaces*

This is connected to the requirements of the subsystems and what context related information of the main GUI they need, for example "current experiment".

Close cooperation is needed with WP3i (Assembling the PRISM System). WP3i will be required to make sure all configuration information can be accessed from the GUI and to provide for the scripts used to run the models. These scripts typically deals with the:

- Creation of executables.
- Hardware specification (model resolution, number of processes used....)
- Input/Output.
- Initialisation of physical parameters and physical constants.
- Distribution of the different type of files into the tree architecture.

## *Software life cycle management*

Initially manual administration will have to be considered. Typically an administrator has to log into the site. Tools such as Secure Shell should make this possible. Semiautomatic administration should be developed allowing for administrators to install new versions of models and for infrastructure components to be updated efficiently.

Component models should be updated through the administration interface. This interface is the composition GUI used for experiments but will now enable the administrator to enter locations for files, change compilation options and the like. These responsible components are number 1, 9 and 22 in Table 21. A typical administration scenario is described:

- Administrator starts GUI.
- Loads configuration of site X component model M1.
- Enters the location of the sources as ftp address.
- Changes other settings and gives new version number.
- Presses submit.
- A build job is created at site X with the configuration and queued for execution.

The outline above suggests that this procedure can only succeed by using a high standardization of build scripts, makefiles, versions and source layout. This standard needs to be described, defined and followed by the model developers to allow us to create an automated procedure.

For infrastructure components such as the Request Listener (RL) and the administration server, a semiautomatic system should be developed. Components consisting of archive files with java classes and associated configuration data can be triggered for automatic installation on a site. The viability of this solution depends on the security model implemented. The component responsible for this is no 23 in Table 21.

### *Access Control*

The area of access control has not been addressed by any other document and as such is undefined for archiving, visualization, monitoring and running experiments.

This question is related to the security implementation.

It is expected that unix permissions related to user ids will be implemented and that only pre registered per user access to PRISM sites will be allowed.

## III.9 Implementation Plan, Deliverables and Desirables

In general the implementation should be feature based, i.e. we first deliver a very basic but useful functionality. Subsequent development is concerned with developing new functionality as "features" with each new feature requiring a new release. The implementation is to be made as a local system and a WS system.

### *Description of Local System*

The local system will run the server and client parts as one component on a local computer. The software will be made available from a repository at <http://prism.enes.org/>. The software will deliver a GUI run from the command line that reads all configuration files from the current working directory. The user will be in full control of all configuration files and can edit them at will. This system is aimed primarily at model developers (local system).

### *Description of WS System*

The WS system involves splitting the local system into its separate software components and moving the configuration files developed with the local system to the server. This requires the security mechanisms to be in place.

### *Security Implementation*

The security requirements have been described in detail in the REDOC II.1 part of the document (page 43). Some implementation aspects are:

- The modularity of the solution.
- The time scale of implementation.
- The use of existing software independent products.
- Discovery and implementation of new schemes in the computing community.

One of the reasons that the security implementation is difficult is that there are currently no ideal security implementations available that are scalable, cheap, secure and convenient. New technologies are being held back because of this and there is a lot of activity in the field to overcome this situation. Standardization and reference implementations can be expected within the progress of the PRISM project. Further, there are a variety of solutions in place at different PRISM sites and a survey is made to gather information on their preferences.

The security solution involves authentication and security between computers and of humans communicating with computers.

From the implementation plan we can see that a security implementation will not be required until month 24. It will be a further year until a fully service based system is in operation and advanced service to service authentication needs to be in place. Therefore the final security solution is kept open to allow for the use of new emerging standards and initially concentrate on modular solutions with low maintenance requirements.

### *Proposed solution*

For authentication between humans and computers we should use a software independent simple password related system such as s/key. This system can be implemented with HTML and Java based solutions and is well known to system administrators who already maintain password files. Implementations are freely available. Communication can be secured by signatures but should not be encrypted.

For authentication between computers standardization is currently under way by the Sun Web Messages Security API (<http://jcp.org/jsr/detail/183.jsp>) and also by World Wide Web Consortium Encryption WG (<http://www.w3c.org/Encryption/2001/>) and other related XML technologies such as SAML (<http://www.oasis-open.org/committees/security/>).

The basic of these technologies is to use signatures to secure the messages from being tampered with and public key encryption to allow for authentication. An implementation building on the password files as a shared secret key is possible and could be implemented if no standards have emerged.

### *Plan*

The implementation plan builds on creating the necessary infrastructure for the WS system on the central site. Parallel to this the local system is used by developers to work on their models so that they can be made available to the central system.

Once the quality and operation of the system is satisfactory the remote servers on other sites will be developed and deployed letting users submit experiments on other sites.

This has the advantage of not spending administration efforts on inferior quality software and time on remote debugging sessions.

## *Phases*

The implementation plan consists of 4 different phases: Foundations, prototype, network and integration, documentation and features. The plan tries to move the features not necessary for basic operation to the later phases.

It is important to determine the critical path in the development and the functionality that can stop the progress of the project.

### *Critical Path*

It is essential to determine the critical path for the implementation of components. Dependencies are given for each phase.

#### *Month 6 to 12: Foundations, Local System*

- Administration scripts
- Build administration interface
- Adapt existing GUI
- Tasks: 1, 7, 21, 22 of Table 21

**Deliverable:** Delivery of local system to modellers.

**Dependencies:** Deployment scripts defined and available, deployment structure agreed on, XML formats known and defined with examples, standard build scripts defined and created.

#### *Month 12 - 18 Prototype of Web-based System*

- Develop monitoring server
- Develop administration server
- Tasks: 12, 9, 6 of Table 21
- Additional work is required to set up a central site.

**Deliverable:** Prototype of web based system.

**Dependencies:** Standardized configurations made available by modellers, monitoring tool available.

#### *Month 18 - 24 Testing, Security and Feature Development*

- Develop/Test security
- Development of network related infrastructure
- ORB, Directory server.
- Tasks: 3,16,17,20 of Table 21

**Deliverable:** Deliver system with components installed in multiple locations.

**Dependencies:** Security and access model defined, test systems available, archiving of data must be working and accessible or results will have to be deleted.

### *Month 24 - 36 Network and Integration of Visualization and Diagnostics Tool*

- Event service
- Integration of visualization and diagnostics.
- Semiautomatic software distribution system.
- Documentation
- Tasks: 8,18,19,23 of Table 21

**Deliverable:** Deliver fully tested and documented system.

**Dependencies:** Security implemented, interfaces to visualization and diagnostics tools implemented and available for integration.

## III.10 Integration with other Projects

The fact that other projects with similar goals (Globus etc.) are emerging and being developed raises the question of integration and also cooperation. If a high degree of inter operability is required we should test their systems and also make contact to discuss how we can both benefit from each other's development. Advantages could be reuse of software but the time it takes to identify these areas is at stake.

### III.11 Risks

Type	Magnitude	Description	Impact
Security demands incompatible on some sites.	Severe	Multiple security solutions may be necessary.	If sites cannot agree on one security solution it may introduce costly separate solution affecting the client experience.
Deployment and administration cycles made difficult due to security.	Severe	Increased manual intervention necessary at higher cost.	Synchronization of software is delayed.
Hardware resources unavailable for testing and implementing the infrastructure.	Fatal	System cannot be built and tested.	No system.
Standardization of build-scripts not achieved.	Severe	Models not conforming to standards.	Manual intervention required.
Creation of configuration information for GUI incomplete.	Severe	Necessary information missing for configuration or deployment.	GUI will not work properly and automation benefits lost.
Deployment standards not achieved.	Severe	Deployment not achieve standards if validation and editing tools not	Deployment will fail.

Table 23: Risks

## IV - System Components

### IV.1 PRISM Coupler and I/O Library

*The coupler drives the whole coupled model, ensuring the synchronization of the different component models and the exchange of the coupling fields directly between the components or via additional coupling processes. When needed, the coupler performs transformations on the coupling fields. Another important part of the coupler is the model interface library linked to each component model, which interfaces it to the rest of the coupled model. As I/O and coupling data share many characteristics, it was decided to develop one common model library for both purposes.*

*The different constituents of the PRISM coupler and I/O library are therefore the Driver, the Transformer, and the PRISM System Model Interface Library (PSMILe). The PSMILe includes the Data Exchange Library, which performs the exchanges of coupling data, the I/O library, and some coherence check and local transformation routines.*

*In the first section the PRISM coupled model high-level architecture is first presented. The functionalities of each constituent and their priority of development are detailed in the second section.*

---

#### IV.1.1 Coupled Model High Level Architecture

The elements of a coupled model are the following:

- The **Driver**, which monitors the whole coupled model.
- The Transformer separate entity **T**, which performs transformations on the data.
- The component models **M<sub>i</sub>**, **M<sub>j</sub>**, or **M<sub>k</sub>**, interfaced to the rest of the coupled model through the PRISM System Model Interface Library (dark and light blue squares).
- The Potential Model Input and Output Description (**PMIOD**): a container describing the relations the model is able to establish with the rest of the coupled model through inputs and outputs.
- The Specific Model Input and Output configuration (**SMIOC**): a container describing the relations the model will establish with the rest of the coupled model through inputs and outputs for a specific experiment.
- The Specific Coupling Configuration (**SCC**): a container describing all activated coupling fields for a specific experiment.
- The files, containing data.

An overview of a coupled model is presented in Figure 29.

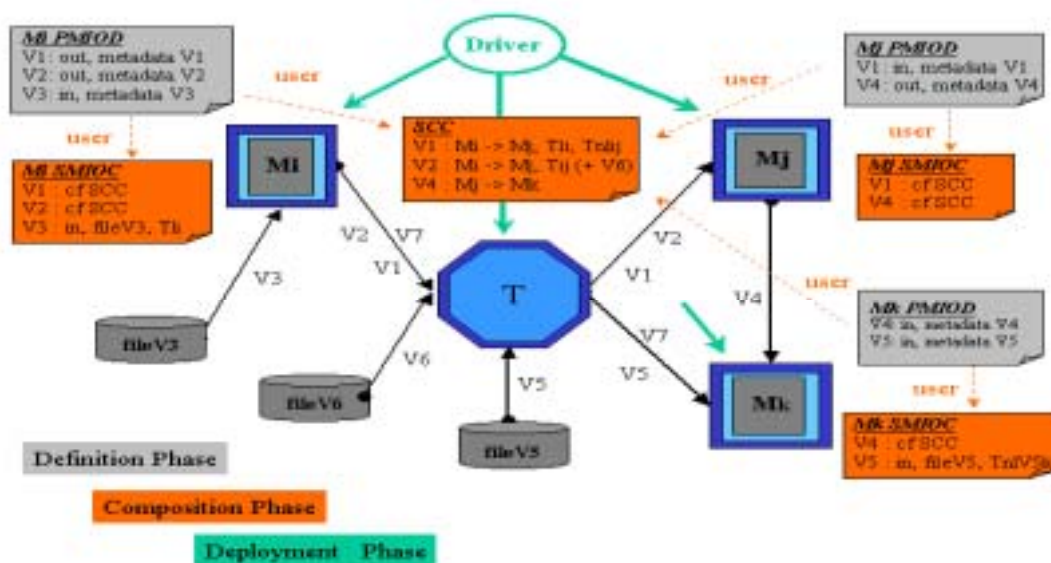


Figure 29: Details of the different parts of the coupled PRISM model

As detailed in the REDOC II.2, page 59, I/O data, i.e. data coming from or going to disk, and coupling data, i.e. data coming from or going to another model, share many characteristics, and it was therefore decided to develop one common model library for both purposes. Both types of data are concerned by the present high-level architecture.

The different elements of a coupled model are detailed hereafter by describing the three basic phases of its construction and execution:

- **Definition** of the entities to be coupled (component models, coupler elements, files...)
- **Composition** of these entities in a coupled system
- **Deployment** of the coupled system onto a set of computing resources

## A - Definition phase

In the definition phase, the different elements of the coupled system are prepared:

- The component models (Mi, Mj, or Mk), including the PSMILe:

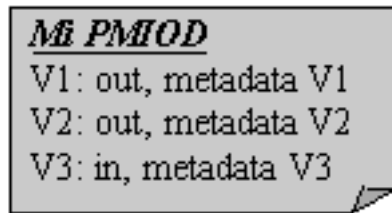


Each component model has to include specific PSMILe instructions that will allow the component model to interact with the rest of the PRISM System at run-time. The PSMILe, represented here by the dark and light blue squares, includes the Data Exchange Library, which performs the exchanges of coupling data directly between the component models



or between the component models and other coupling processes, the I/O library, and some coherence check and local transformation routines.

- The Potential Model Input and Output Description (PMIOD):



For each model, the Potential Model Input and Output Description (PMIOD) describes the relations the model is able to establish with its external environment. The PMIOD contains a short description of the model, its grids, and the list of all data requested or produced by that particular component model and their description (the meta-data). The input and output data can be divided into 3 categories.

- **Transient input and output variables:** data evolving during the run, received or provided at run-time by the model at an a priori unknown frequency, from or to an external entity (another model or a disk file). For example, coupling data and diagnostics belong to this category.
- **Restart variables:** data requested initially by the model to (re-) start the simulation and provided previously by the model itself (not by an external entity). These data are saved to disk at regular intervals during each run. It remains to be clarified if restart variables should be treated differently from transient input and output variables.
- **Persistent input parameters:** data requested initially by the model to define the physical configuration of the experiment. These parameters may have default values but may be adapted by the user. For coupled models, the "**universal parameters**" are parameters that require a consistent definition among all the model components.

The model administrator makes each model and its respective PMIOD available to potential PRISM users.

- The input files:



All input files containing data required for the run have to be generated.

- The coupler Driver and Transformer separate entity:

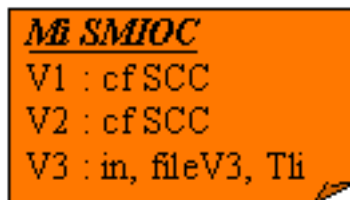


The Driver, which monitors the whole coupled simulation, and the Transformer separate entity, which performs required transformation on the data, have to be available.

## B - Composition phase

In the composition phase, a particular user assembles a particular coupled model.

- Selection of component models:  
The user first chooses the component models he wants to couple for one particular experiment.
- Input file selections:  
The user selects the input files containing information that will be used during the simulation, such as forcing fields.
- Driver and Transformer separate entity selection:  
The user selects the PRISM Driver and Transformer separate entity.
- Constitution of each model Specific Model Input and Output Configuration (SMIOC):



MI SMIOC  
V1 : cf SCC  
V2 : cf SCC  
V3 : in, fileV3, Tli

The diagram shows a rectangular box with an orange background and a black border. Inside the box, the text is as follows: MI SMIOC, V1 : cf SCC, V2 : cf SCC, and V3 : in, fileV3, Tli. The box has a small folded corner effect at the bottom right.

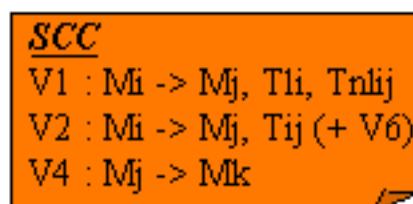
Based on each model PMIOD, the user generates for each model an SMIOC. The SMIOC describes the relations the model will effectively have with its external environment through inputs and outputs for a specific experiment.

For **transient input and output variables**, the user may decide that a particular data will 1- have no role in the simulation, 2- be read from a file or written to a file (I/O data), or 3 - be exchanged between to component models (coupling data). For I/O data, the user indicates in the SMIOC, the name(s) of the respective file(s), the input or output frequency, and possibly the local and non-local transformations required on the data. For coupling data, the user just refers to the Specific Coupling Configuration (SCC).

For **restart variables**, the user is only allowed to indicate the name of the restart file and, possibly, the restart saving frequency. However, this last parameter should have the same value for all component models and should therefore be treated as a **universal parameter** in the SCC.

The values of **persistent input parameters** are read at run-time in the SMIOC. The user may be allowed to change the default value therein. For persistent input parameters that are also universal parameters, the value taken into consideration is the one indicated in the SCC.

- Constitution of the Specific Coupling Configuration (SCC):



SCC  
V1 : Mi -> Mj, Tli, Tnlj  
V2 : Mi -> Mj, Tij (+ V6)  
V4 : Mj -> Mk

The diagram shows a rectangular box with an orange background and a black border. Inside the box, the text is as follows: SCC, V1 : Mi -> Mj, Tli, Tnlj, V2 : Mi -> Mj, Tij (+ V6), and V4 : Mj -> Mk. The box has a small folded corner effect at the bottom right.

The user constitutes only one Specific Coupling Configuration (SCC) for each particular coupled model simulation. The SCC centralizes the description of all activated coupling fields and all related coupling parameters chosen by the user (source and target models, coupling frequencies, local and non-local transformations, etc.) for one particular experiment. The SCC also contains the **universal parameters** prescribed by the user.

It is proposed that the PMIOD, SMIOC, and the SCC containers be implemented as XLM files.

## C - Deployment phase

At run-time, the different parts of the system will play different roles. A more detailed description of the functionalities of each constituent is presented in section IV.1.2

- The **Driver**: launches the component models, monitors their execution and termination.
- The **Transformer** separate entity T: performs required transformations on the I/O and coupling data.
- The PRISM System Model Interface Library (PSMILe):  
The PSMILe includes the Data Exchange Library, which performs the exchanges of coupling data directly between the component models or between the component models and the separate Transformer entity, the I/O library, and some coherence check and local transformation routines. At run-time, specific PSMILe instructions will perform the following actions:

### Initialization:

- Declaration of PSMILe internal data structure.
- Message passing initialization.
- I/O initialization.
- Initialization of **persistent input parameters**, read directly in the SMIOC.
- Initialization of **universal parameters** either received from the Driver or read directly in the SCC.

### Meta-data declaration and initialization:

- Definition of the meta-data describing input or output data (for example the grid coordinates, mesh areas, mask, partitioning), and definition of associated identifiers.

### Declaration of transient and restart variables:

- Association to the relevant meta-data identifiers (see below).
- Access to user-defined data information: for each data declaration, the PSMILe consults the SMIOC and identifies the user's choice for that particular experiment (coupling or I/O data, input or output frequency, source and target models, source or target file, transformations, etc.)

### **Sending and receiving data**

- The actions performed by the PSMILe below each sending or receiving instruction depend on the user's choices read in the declaration phase in the SMIOC and in the SCC: the library may simply return, or perform local transformations, and/or perform the exchanges between the models, and/or perform the reading or writing into files, etc.

### **Coupling termination**

- All actions related to finalizing the run.

## **IV.1.2 Detailed Functionalities for the PRISM Coupler and I/O Library**

As detailed above, the different constituents of the PRISM coupler are: the Driver, the Transformer, and the PSMILe, linked to the component models and which interfaces the component model with the rest of the coupled model. The PSMILe includes the Data Exchange Library, the I/O library, and some coherence check and local transformation routines.

For each of these constituents, the list of possible requirements established in the REDOC II.2 was revised and choices of functionalities that should be implemented in the different versions of the PRISM coupler were made, considering the answers to the template summarized in REDOC I.3. These choices are detailed below.

For each functionality, a priority of implementation is given: "1" means that the functionality should be provided for the PRISM coupler first version (D3a1, month 12), "2" for PRISM coupler second version to be used in the demonstration runs (D3a2, month 24), and "3" means that the functionality may be provided for the PRISM coupler final version (D3a3, month 36).

### **IV.1.2.1 General Requirements**

- The overhead associated to the global system modularity and flexibility is acceptable. **(2,3)**
- The whole system is portable and efficient on the different hardware architectures used for climate modelling, on dedicated or shared hardware resources. Standard and portable solutions should be preferred. However, for critical issues for which a portable solution would not exist or would lead to very low efficiency, machine dependent options could be offered. **(3)**
- The design and implementation lead to code easy to maintain and can be easily modified to support future model or coupling functionalities. **(2,3)**
- Design reflects a clear separation of responsibilities for the different parts of the coupler. **(2, 3)**
- The PRISM System infrastructure can be used to technically assemble a coupled system based on any component models, even if these models do not conform to the PRISM physical interfaces given that they include the PRISM System Model Interface Library. **(1, 2, 3)**
- The PRISM System infrastructure can be used to couple an arbitrary number of component models; any component can be one-way or two-way coupled with any other component. **(1, 2, 3)**

## IV.1.2.2 Driver functionalities

The Driver manages the whole coupled application. It launches the component models, monitors their execution and termination, centralizes and distributes universal parameters, which require a consistent definition among all component models, and centralizes and distributes information on the component model status during the simulation.

The driver could keep a central role during the whole simulation and manage also the exchanges of coupling data. The preferred design option here is to decentralize the coupling functionalities as much as possible in the Data Exchange Library and in the Transformer, and therefore to reduce as much as possible the role of the Driver. This option is probably applicable only for static coupled simulations and allows an easier evolution toward heterogeneous coupling (different component models running on different machines).

As detailed below, the choice of a *static* Driver was also made. The workload of a static driver is likely to be small, even more if the decentralizing option is followed. The Driver could be one separate process used only for it, but could also sit in one separate coupling process used also for the separate Transformer entity, or even could be part of the PSMILe master process of a master model started by the user initially. The first two options are still open regarding the Driver implementation.

### Model Execution and Control:

- The Driver manages static simulations (with respect to the process management): all component models are launched initially and run for the entire length of simulation. Launching of the executables by the Driver (with MPI2) and initial starting of all components (with MPI1) are supported. **(1, 2, 3)**  
The Driver will not manage dynamic model execution (one or more models starting and ending during the simulation), neither conditional model execution (one model is started during the simulation only if a particular scientific condition is met). WP3h considered the dynamic model execution functionality as essential to run time-slice experiments. Other groups considering this functionality as desirable mentioned that it could be useful to run alternatively different coupling configurations (e.g. for asynchronous coupling). We consider here that chaining different static simulations can fulfil these requirements. Technically, the argumentation presented in REDOC was reviewed and it was evaluated that the disadvantages linked to a dynamic configuration were more important than its advantages. Therefore, the choice was made to design a static, less sophisticated, but more efficient Driver.
- The Driver can start a global coupled system flexible in terms of executables (extreme are: each component is a separate executable, or all components run in parallel or in sequence within only one executable). If the latter mode is chosen for two or more components, the developer will be responsible assembling the components into one executable. **(2, 3)**
- The driver can give some statistic on the load balancing of the run. **(3)**
- The driver includes a timing functionality that can sample with identical absolute time the duration of events for all component models. **(3)**

### Information Management:

- The universal model input parameters are parameters that need to be consistently defined in the coupled system (initial date, length of integration, calendar, earth radius, restart sav-

ing frequency, etc.). The user defines this information in the SCC. The driver gets these parameters in the SCC and, on a PSMILe request, transfers this information to each component model. (For a stand-alone model, the model PSMILe should read directly the information in the SMIOC.) **(1, 2, 3)**

- The driver does not centralize all model information (grid definition, distribution, etc.). Each model PSMILe is responsible for transferring the appropriate information to the appropriate processes. **(2, 3)**
- Depending on a log level chosen by the user in the coupled model configuration file, internal PSMILe log functions or log functions implemented by the developer in the code are activated. The log function transfers information on the state of the model to the driver, which centralizes this information and may then transfer this information to a higher-level controlling layer or to the user. **(3)**

### **Coupling Exchange Management:**

- In a decentralized and static approach, the matching between output coupling data produced from one model and input-coupling data requested by another model could be performed initially and the Data Exchange Library included in each model PSMILe should manage the exchange at run-time.  
The other option is that the driver performs the matching and, at run-time, manages the exchanges of coupling data based on this matching. The decentralized option is the preferred one but the two options are still open now. The matching is based on user's choices indicated in the SCC; the way the matching and the exchanges are managed are, in any case, transparent for the developer and for the user.

### **Termination and Restart:**

- If one component aborts, the whole simulation must shut down cleanly. **(1, 2, 3)**  
The coupled model will most probably be a MPI1 or MPI2 application. In that case, if one component model aborts, the whole MPI job terminates automatically and the Driver can have no control on the termination. However, in the case of PSMILe exceptions (the error code returned by one PSMILe routine indicates an error), the model developer may decide to make the model to abort; the routine called in that case first interacts with the Driver which then performs appropriate actions (such as saving the most recent cached information), sends a message to the model which then aborts, and terminates the coupled application. **(3)**
- The driver constantly updates, by writing in a restart log file or by any other equivalent mean, the last date for which all model restarts were saved. This information is sent to the driver by each model PSMILe. **(2, 3)**
- Neither the driver nor any external controlling instance automatically restarts a coupled system after an unforeseen termination (machine breakdown...). This is not desirable because the diversity of faults is large and in many cases, human intervention is mandatory before the simulation should be restarted (ex: the disk on which the diagnostics are saved is full).
- The driver will not be able to shutdown the simulation cleanly if a specific scientific condition is met (e.g. average SST exceeds some predefined value). This functionality will be fulfilled by a conditional PRISM\_Abort implemented in the code by the model developer or by a conditional PRISM\_Abort managed automatically by the PSMILe (see below).

### IV.1.2.3 Transformer functionalities and parallelization

This paragraph first gives some definitions. In the second section, the preferred design options for the PRISM coupler Transformer location and parallelization are presented. In the third section, an exhaustive list of transformations and grids on which these transformations should be performed is presented, together with other specific requirements, and associated priority and calendar.

#### Definitions

- **Point-wise transformation:** an operation that can be completed on each grid point without any external information, neither from the model neighbouring grid points, nor from another model, such as time averaging or addition of coupling fields given on the same grid
- **Local transformation:** an operation that can be completed in a model without any information from another model, such as finding the maximum value of a field.
- **Non-local transformation:** an operation that requires information from another model, such as interpolation.

#### Preferred Design Options for Transformer Location and Parallelization

##### Location for the different types of transformations

The preferred design option is the one in which non-local transformations are performed in the separate Transformer entity (T), as they require information coming from different models. Point-wise and local transformations will be workable in the PSMILe linked to the model before sending or after receiving the data. However, point-wise and local transformations will also be available in the separate Transformer entity T, for example, to combine coupling fields coming from different source models after their interpolation on the target grid.

The same rules apply for two component models assembled into one executable: all point-wise and local transformations will be performed directly in the PSMILe, while the data will have to be treated by the separate Transformer entity T if non-local transformations are required. This last case however is not likely to happen, as two components assembled into one executable will in most cases share the same grid and same partitioning.

Ideally, the coupler should decide the choice of whether the transformation is performed by the PSMILe or in the separate Transformer entity T automatically and this should be transparent for the user (3).

##### Transformer parallelization

As detailed above, the transformation routines included in the PSMILe will perform local transformations, and not only point-wise transformations; their full parallelization is therefore required when the PSMILe is linked to a fully parallel component model (3).

Non-local transformations will be performed in the separate Transformer entity T. Different options of parallelization are possible. The "one-executable full parallelization" option presented in REDOC on page 69 is the preferred one (3). A fall back solution would be a simpler parallelization of the separate Transformer entity T as one executable with openMP.

## *List of transformations, grids, and associated priority and calendar*

### **List of transformations**

A list of relevant transformations is given hereafter. For each transformation, it is specified whether the transformation is "point-wise", "local" or "non-local".

- 1D, 2D, and 3D spatial interpolations  
All these transformations are non-local.
  - S1 - Nearest-neighbour interpolation function:  
For each target grid point, the n nearest neighbours on the source grid, weighted or not by their distance, are averaged.
  - S2 - Nearest-neighbour Gaussian weighted interpolation function:  
For each target grid point, the n nearest neighbours on the source grid, weighted by the value of a Gaussian function at their distance from the target point are averaged.
  - S3 - 1st order interpolation function:  
Standard 1st order linear interpolation.
  - S4 - 2nd order interpolation function:  
Standard 2nd order cubic interpolation.
  - S5 - First order conservative remapping:  
This scheme will guarantee that the line (1D)-/area (2D)-/ volume (3D)-integrated field (e.g. water or heat flux) is conserved between the source and the target grid.
  - S6 - Second order conservative remapping:  
This scheme will also guarantee that the line (1D)-/area (2D)-/ volume (3D)-integrated field (e.g. water or heat flux) is conserved between the source and the target grid.
  - S7 - Remapping using user-defined remapping info (ex: MOZAIC)
- Other 1D, 2D, and 3D spatial transformations
  - S8- Conservation:  
This non-local operation ensures global energy conservation between source and target grids (ex: CONSERV).
  - S9- Combination or merge:  
This local and point-wise operation combines different parts of different coupling fields or of other predefined external data given on the same grid (ex: FILLING). This operation may involve the smoothing of the fields near the different domain borders; in that case, the operation is still local but not point-wise.
  - S10- Masking:  
With this local and point-wise operation, only the points listed in index have meaningful data and the others are changed to a missing value (ex: MASK).
  - S11- Scattering:  
This local operation scatters the model data onto the points listed in an index.
  - S12- Gathering:  
This local operation gathers from the input data all the points listed in an index.
  - S13- Collapse:  
This local operation results in the collapse of any dimension or combination of dimensions by various, possibly weighted, statistical operations, such as mean, max, min,



etc. (possibly relative to a threshold, e.g. maximum of positive values). (Ex: CHECKIN, CHECKOUT)

- S14- Subspace:  
This local operation results in the extraction of subspaces or hyperslabs in any combination of spatial dimension.
- S15- Algebraic operations:  
Local and point-wise operations, such as addition, subtraction, multiplication, etc., with possibly different coupling fields or predefined external data (given on the same grid) and numbers as operands (+, -, X, SQRT, ^2, SIN, LOG...) (Ex: BLASOLD, BLASNEW, SUBGRID, CORRECT)
- S16a - 1st order extrapolation function
- S16b - 2nd order extrapolation function:  
This transformation is required to address the specific problem of the wind curl along the coast.
- Time operations
  - T1- Time integration, average, variance, extrema, linear interpolation  
Local and point-wise operations.

## List of grids

The following grids should be supported for the above scheme. These grids have the following common characteristics:

- The grids may have masked grid points.
- The grids may have "holes" (i.e. they do not cover the whole sphere, e.g. regional grids).
- The grids may be global or regional (except H3 reduced grid which is always global).
- The grids may have overlapping grid points (except H3).

## 2D grids

- H1 - lat-lon grids:  
The grid is given by the intersection of meridians and parallels. Each (i,j) grid point can be described by the value for the indices i and j of two 1-D arrays, latitude(j) and longitude(i).
  - The latitudinal mesh sizes can be regular or irregular, i.e. latitude(j) is or is not constant.
  - The longitudinal mesh size can be regular or irregular, i.e. longitude(i) is or is not constant.
  - The grid may overlap in longitude with N overlapping grid points.
  - The grid may have grid points at the pole and/or at the equator.
  - The border of the cells should be given with the grid.
- H2 - Cartesian and stretched and/or rotated grids (logically rectangular):  
Each (i,j) grid point can be described by the value for the indices i and j of two 2-D arrays, latitude(i,j) and longitude(i,j).
  - The grid may overlap itself in the i and/or j direction.

- The exact location of the mesh borders should be given with the grid.
- H3 - Reduced grids:  
The grid is composed of a certain number of latitude circles, each one being divided into a varying number of longitudinal segments. The grid can be described by the number of latitude circles,  $N_{lat}$ , the latitudinal position of each circle  $n_{pos}(N_{lat})$  and by an 1-D array giving the number of longitudinal segments for each latitude,  $n_{seg}(N_{lat})$ . The total number of grid points,  $N_{tot}$ , is the sum of all  $n_{seg}$ . The grid can also be described by two 1-D arrays,  $latitude(N_{tot})$  and  $longitude(N_{tot})$ . This grid may be considered as one particular case of unstructured grids
  - There is no overlap of the grid.
  - There is no grid point at the equator nor at the poles.
  - There are grid points on the Greenwich meridian.
  - The exact location of the border meshes should be given with the grid.
- H4 - Unstructured grids:  
The grid, with  $N_{tot}$  number of grid points, has no logical structure. The grid must be described by two 1-D arrays,  $latitude(N_{tot})$  and  $longitude(N_{tot})$ 
  - The grid may overlap itself for some grid points.
  - There may be a grid point at the equator or at the poles.
  - There may be grid points on the Greenwich meridian.
  - The mesh can have an arbitrary number of sides. The exact location of the border meshes should be given with the grid.

### 3D grids

- V1 - Reproduction of the same horizontal grid at different levels:  
The same horizontal grid is reproduced at different vertical levels. Each level has its particular mask.  
The vertical levels can be:
  - V1-1 : given at regular or irregular depth or height levels (z co-ordinate)
  - V1-2: hybrid: first level follows the orography (atmosphere models) or the bathymetry (ocean models), last level follows an isobar (atmosphere) or the surface (ocean), progressive transition in between.
  - V1-3 : given at regular or irregular isopycnal (density) levels (r co-ordinate)
- V2 - Different horizontal grids at different levels:  
The horizontal grid is not reproduced at different vertical levels. The horizontal grid can be rotated, translated, or totally unstructured.

### Other specific requirements

- To support scalar coupling data. (For priority see below).
- To support vector coupling data in the standard spherical geographical coordinate system. (For priority see below).
- To support vector coupling data in any set of local coordinate system. (For priority see below).

- To support fields with undefined variables. **(2, 3)**
- To support coupling fields whose characteristics may change over time as the simulation develops , e.g. grid, resolution or distribution (CF dynamic simulation). **(3)**
- To be able to save, at a user-defined frequency, its restart data (e.g. time accumulated data). **(2,3)**
- To support source and target coupling domains that totally or partially overlap (e.g. global atmosphere with a regional ocean, regional model nested into global model, etc.). **(1, 2, 3)**

### **Priority and Calendar**

The following paragraph gives the priority of development. The meaning of the numbers **1, 2, or 3** is given above for the different transformations on the different grids listed there. When two numbers are given, parts of the functionality will be provided for the respective coupler versions.

## Transformations on 2D Vector Coupling Fields

	H1 - lat-lon	H2 - log. rect.	H3 - reduced	H4 - unstructured
S1 - near.neigh	1	1	1	1
S2 - Gaussian	1	1	1	1
S3 - 1st O interp.	1	1	3	-
S4 - 2nd O interp.	1	1	3	-
S5 - 1st O cons rem	1	1	1	1
S6 - 2nd O cons rem	3	3	3	3
S7 - user remapping	1	1	1	1
S8 - conservation	1	1	1	1
S9 - combination	1	2	2-3	2-3
S10 - masking	1	1	1	1
S11 - scattering	2	2	2	2
S12 - gathering	2	2	2	2
S13 - collapse	2	2	2-3	2-3
S14 - subspace	2	2	3	3
S15 - algebra	1-2	1-2	1-2	1-2
S16a - 1st O extrap.	1	1	1	-
S16b - 2nd O extrap.	2	2	3	-
T1 - time operation	2-3	2-3	2-3	2-3

Table 24: Transformations on 2D scalar coupling fields

- Transformations on 2D vector coupling fields given in the standard spherical coordinate system.  
Transformations S1, S2, S3, S4, S6, S7, S9, S10, S11, S12, S13, S14, S15, S16a, S16b, T1 are given the same priority than for 2D scalar fields. Transformations S5, S8 are of priority 3.
- Transformations on 2D vector coupling fields given in any set of local coordinate system:  
Transformations S1, S2, S3, S4 for H1 and H2, S7, S10, S11, S12, S15, S16a, S16b for H1 and H2 are of priority 2. Transformations S4 for H3, S5, S6, S8, S9, S13, and S14, S16b for H3 are of priority 3.

## Transformations on 3D Coupling Fields

- Interpolations for V1-1 and V1-2 grids  
For V1-1 and V1-2 grids, the treatment of 3D coupling fields will be addressed for the second PRISM coupler version. The idea is to proceed with a multiple 2D interpolation (i.e. 2D interpolation on many horizontal or hybrid levels) plus a simple linear interpolation vertically (priority 2). The priority given to the multiple 2D interpolations or extrapolations (S1, S2, S3, S4, S5, S6, S7, S16a and S16b) for scalar or vector fields for the different horizontal grids is 2, or more if a lowest priority is given above for the equivalent (single) 2D interpolation scheme.
- Other transformations for V1-1 and V1-2  
For V1-1 and V1-2 grids, the local and point-wise transformations S10, S15, and T1 are given priority 2. For V1-1 and V1-2 grids, transformations S8, S9, S11, S12, S13, S14 are given priority 3.
- Interpolations and other transformations for V1-3 and V2 grids

These transformations will not be addressed within PRISM 3-year project.

## *Transformations on 1D Coupling Fields*

All transformations are given priority 3

### IV.1.2.4 PSMILe Functionalities

The PSMILe is the set of routines implemented in a component model code to interface it with the rest of the coupled model. The classes of PSMILe instructions that will be invoked in the component model code at run-time are described in the section on the “Deployment phase” on page 139. Here, the functionalities of the different PSMILe constituents (i.e. the Data Exchange Library, the I/O library, and the coherence check and local transformation routines) are presented in more details.

#### *PSMILe General Characteristics*

- The modifications to implement in the model code are as reduced as possible. **(1, 2, 3)**
- The PSMILe is layered, and complexity is hidden from the component code. **(1, 2, 3)**
- Interface routines once defined and implemented are not (or as little as possible) subject to modifications between the different versions of the PRISM coupler. However new routines may be added. **(1, 2, 3)**
- A good trade-off is chosen between (I) a concise list of parameters for each subroutine call (more subroutines provided with a shorter list of parameters) and (II) a small number of subroutines, each one having a longer and more complex parameter list. The complexity arises from the need to transfer not only the coupling data but also the meta-data. **(1, 2, 3)**
- The component models are able to run in a stand-alone model without modifications, with or without an external driver. **(2, 3)**
- The description of the data, i.e. the meta-data (e.g. units, grid coordinates, mask, distribution, ...), and the model information (e.g. length of a time-step) is given by the model through the PSMILe and not duplicated externally by the user **(2, 3)**. This information may change during the simulation. **(3)**
- The PSMILe is extendable to new types of coupling data (e.g. data given on arbitrary grids). **(1, 2, 3)**
- The PSMILe includes the Data Exchange Library and the I/O library as the most external layers. The PSMILe therefore automatically manages the cases for which input coupling data are not provided by another model but have to be read into a file; this is transparent for the component model. The format of these data files could be a standard PRISM fixed format **(2)**. At a later stage, different formats could be supported for these data files, implying that the I/O instance can interpret their content. **(3)**
- The PSMILe includes some local transformation routines and performs the transformation required locally before the exchange with the rest of the PRISM System. **(2, 3)**
- The PSMILe performs some checks of coherence on coupling and I/O data, according to a coherence check level defined by the user in the coupled model configuration file. **(3)**

## *Data Exchange Library (DEL) Functionalities*

The Data Exchange Library (DEL) performs the exchanges of coupling data between the component models, or between the component models and the separate transformation entity. The DEL must therefore be included as the most external layer in the PSMILe.

Data transfer between separate processes will be implemented using the message-passing interface MPI, which is a widely used and portable standard. MPI implementations completely supporting the MPI standard are available for every architecture used by the climate modelling community either as open source public domain code or as proprietary software optimized and installed on high performance computer system. Furthermore MPI is best suited for the close coupling between separate processes, as in climate system modelling, since individual MPI implementations are designed to use the most efficient network on a specific architecture.

Since all parallel climate model codes support communication via MPI the introduction of alternative approaches like CORBA requires additional software like Fortran ORBs. Another possibility is wrapping the Fortran codes using a C++ ORB, which can require major changes to the involved Fortran codes as well. (For experiences gained with wrapping Fortran code see [http://accl.grc.nasa.gov/IPG/CORBA/wrap\\_fortran.html](http://accl.grc.nasa.gov/IPG/CORBA/wrap_fortran.html)).

In addition, alternative approaches such as CORBA handle data transfer via TCP/IP, which is not well suited for a fast and efficient parallel data transfer. MPI processes may communicate simultaneously without interfering the communication of other processes, while the same kind of communication will cause conflicts on a TCP/IP connection. Transfer rates between two processes can differ by a factor  $10^5$  to  $10^6$  when comparing CORBA with MPI. Furthermore a complete CORBA standard is not available for every architecture.

The DEL detailed functionalities are:

- The exchange can occur directly between two component models without going through additional transformation processes. When the component models are parallel and have different data partitioning, repartitioning associated to direct communication is required; all type of distributions usually used in model component codes are be supported. **(2 for static simulations, 3 for CF dynamic simulations)**
- "End-point" data exchange: when producing coupling data, the source model does not know what other model will consume it; when asking for coupling data a target model does not know what other model produces it. **(1, 2, 3)**
- Coupling data produced by a source model can be consumed by more than one target model. **(2, 3)**
- Occurrence of the exchange can be different for the different coupling fields. **(1, 2, 3)**
- For each coupling field, the exchange occurs at a fixed frequency for the whole simulation. **(1, 2, 3)**
- For each coupling field, the exchange may occur at different fixed frequencies for different periods. **(2, 3)**
- Coupling data produced from one model at a particular time may be required as input coupling data for another model at another time. The DEL can take into account a time lag defined by the user in the coupled model configuration file. **(2, 3)**
- Conditional occurrence of an exchange, depending on parameters dynamically calculated during the simulation, will not be supported within PRISM 3 years.

- The coupling data can be of different types: integer and real, complex (32, 64, 128 bits) appearing as multidimensional arrays, but without time dimension **(2,3)**. For structures, operators and functions, other PSMILe primitives will be defined when required, but not within PRISM 3 years.
- The coupling data characteristics, and therefore the associated meta-data, may change over time as the simulation develops e.g. the grid or the resolution (CF dynamic simulations). **(3)**
- Coupling data produced by one model may be only partially consumed by the target model; extraction of subspaces, hyperslabs or indexed grid points may be required before the exchange. **(2,3)**
- Sending and receiving instructions can be placed anywhere in the source and target code and possibly at different location for the different coupling fields. **(1, 2, 3)**
- The DEL insures that component models can be executed concurrently, in a regular sequence (one after the other), or in some pre-defined combination of these two modes. **(1, 2, 3)**
- The DEL offers efficient data exchange implementations for loose and strong coupling. Loose coupling is the configuration in which the two component models are run sequentially or concurrently as two separate executables. Strong coupling is the configuration in which the two component models are run within the same executable. **(2, 3)**
- The DEL could perform the matching between output coupling data produced by one model and input coupling data requested by another model (see related discussion above in the Driver section) **(1, 2, 3)**

## *I/O Library*

The I/O library performs the exchanges with files stored on disk. The user selects activated variables, regions, temporal and geographical transformations and file names are chosen in the SMIOC (see Section “Composition phase” on page 138). Meta-data and run time information are provided at run-time by the component model through the PSMILe. The data and the associated meta-data will be read or written to the disk files.

For data access, calls to the NetCDF (<http://www.unidata.ucar.edu/packages/netcdf/>) library will be implemented. Support of formats other than NetCDF will not be implemented, but entry points for reading and writing other file formats will be provided.

Execution on parallel machines will have to work efficiently. MPI-IO is the standard solution and will be evaluated. In a first step, we will avoid parallel I/O by doing regional selection for input data and by doing post-processing operation after a simulation to combine multiple outputs files provided by a parallel execution.

## *Coherence check routines*

The PSMILe will perform some checks of coherence on coupling and I/O data, according to a coherence check level defined by the user in the Specific Coupling Configuration (SCC) file. The coherence check instance will:

- Understand some standard conventions of meta-data. **(3)**

- Based on the meta-data, perform compatibility checks between data produced by a source component and its description established by the model developer in the PMIOD. **(3)**
- Based on meta-data description, perform compatibility checks between data produced by a source component and data required by a target component. **(3)**
- Based on meta-data description, recognize type of coupling data (flux, vector, scalar) and verify that the user's transformation choice is appropriate. **(3)**
- Warn the user if the coupling and I/O frequencies are not synchronized. **(3)**
- Automatically check an abort condition defined by the user in the SCC or in the SMIOC on the value of the input and output data, and perform the abort if the condition is met. **(3)**

### *Local Transformation Routines*

The PSMILe will perform the following transformations locally. The priority for including these local transformations in the PSMILe is given here:

- Combination of different data produced by one model (S9) **(2)**.
- Masking (S10) **(2)**.
- Scattering (S11) **(3)**.
- Gathering (S12) **(3)**.
- Collapsing (S13) **(3)**.
- Subspace (S14) **(3)**.
- Algebraic operations (S15) **(2)**.
- 1st and 2nd order extrapolation (S16a, S16b) **(3)**.
- Time integration, average, variance, extremes and linear interpolation (T1) **(2)**.



## IV.2 Diagnostics and Visualization

*The Archive, Data Processing and Visualization architectural design choice is specified. The meta-data standard chosen for the output of PRISM model data and for data exchange is the CF convention. NetCDF will be the supported file format for both model output and data exchange. An initial prioritized list of functionalities required for the processing library and visualization system has been constructed, with input from the PRISM community. A review of existing graphics packages is currently underway.*

---

### IV.2.1 Introduction

Within PRISM a flexible library of tools will be built to facilitate processing and analysis of data in the common PRISM data format and to promote sharing of data and analysis programs. High-End and Low-End graphical interfaces will be developed to display the data. This document details the architectural design choices made for the Data Processing library, visualization and archiving, including views from the PRISM community.

### IV.2.2 Meta-Data and File Format Definitions for Model Output and Data Exchange

#### *Meta-Data Convention*

Meta-data allows us to fully describe geophysical data. This is valid for data, which is available to the user in files as well as for data that is exchanged between models or different software packages. It is important to note that up until now models or software libraries did not exchange this type of information and that it was up to the user to make sure that the correct variables were passed. With a proper meta-data convention we can ensure that the models verify the data they receive. This is an essential step for PRISM, which aims at coupling models from different origins and by people who have not necessarily been involved in the development of these models.

The CF convention (<http://www.cgd.ucar.edu/cms/eaton/netcdf/CF-20010808.htm>) is proposed as the meta-data standard for PRISM.

- CF is an extension of the existing standard COARDS. (For information on COARDS see [http://ferret.wrc.noaa.gov/noaa\\_coop/coop\\_cdf\\_profile.html](http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html))
- CF meets the meta-data requirements of PRISM
  - The CF convention provides meta-data to describe source, history, institution, etc. It should, consequently, be possible to have variables with different source, institution, history, etc. in the same file.
  - CF Meta-data is intended to describe physical data at points or in cells. PRISM files may contain other kinds of information, which would need different kinds of meta-data to describe them. For example, it may be required that a formula used to calculate temperature at a certain level in the atmosphere model may need to be passed to the ocean model. The PRISM meta-data definition needs to be extendable beyond the CF

convention to cater for this type of data transfer. Multiple files may be required to cater for data belonging to different conventions, but should be treated as a single logical file.

- PRISM will need to agree lists of standard names for data fields and co-ordinates. CF includes a standard name table; the requirements of PRISM can be incorporated into this.
- CF was designed from the ground up with Climate Prediction needs in mind and was developed by international collaboration (Eaton, Gregory, Drach, Taylor and Hankin). It was developed over a long period with input from many people.

A couple of extensions to the CF convention will be made in order for CF to fully meet our requirements:

1. The meta-data needs to record extra information for a full description of the extent and shape of cells in non-rectilinear grids, for instance cell area. A general method is being added to CF in order to support this.
2. The meta-data should be able to support non-spherical grids e.g. for data on a Cartesian plane. No specific support is offered in CF for this, but nor is it disallowed.
3. For its use in the exchange of data between models the CF convention will need to be extended with attributes related to the numerical properties of the variables. This will ensure that the various models manipulate the data with the right operators and in the right sequence.

It is proposed that there should be a common interface between the model and the coupling and diagnostic systems. This will be simplified by using a common meta-data definition for both purposes.

### *File format*

NetCDF (<http://www.cgd.ucar.edu/cms/eaton/netcdf/CF-20010808.htm>) will be the file format supported for model output and data exchange because it

- Is used by COARDS
- Is used by AMIP
- Is freely available and portable
- Supports the CF convention
- Allows alignment of development effort with internal software development at the Met Office (The COCO project)

While netCDF will be the supported file format it will, of course, be possible to interface to other output file formats. It will then be the responsibility of the institution to provide conversion tools to convert into the common data exchange format.

## IV.2.3 High-level Architecture

The architecture diagram laid out in REDOC II.3 is confirmed, noting that the archive interface will be an extension to the I/O library. Having it as an extension to the I/O library will be more efficient.

Institutions will then plug in their functions to allow data to be output in their format or onto their system.

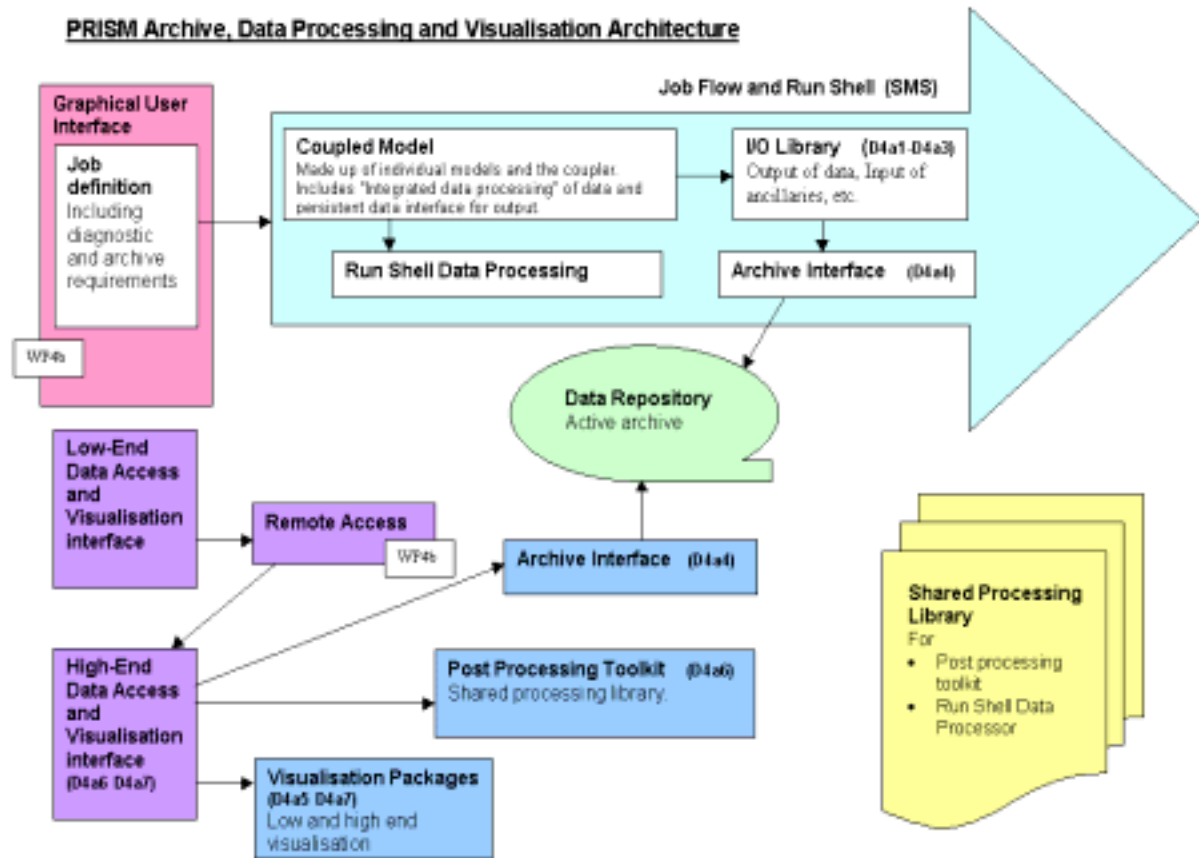


Figure 30: Diagram of the Archive, Data Processing, and Visualization system

### Site Boundaries

- Job Flow and Run Shell, User Interface and Active Archive will all be located on the local site.
- The Data Processing Toolkit and Graphics Package will be located on the remote site.
- The Shared Processing library will be located on both sites to cater for both Low-End and High-End graphics.

### IV.2.4 Processing Library

Requirements for the processing library have been circulated within the PRISM community for acceptance and comments and will be incorporated in this document as soon as available.

PRISM will aim to provide software that will do both processing in bulk (e.g. for post-processing of model output files) and computations on smaller amounts of data for analysis and visualization. The processing library will share code with the I/O library and the coupler.

### *Utilities Required to Support CF*

The Climate Data Management System (CDMS) is specialized for organizing multi-dimensional, gridded data used in climate analysis and simulation. It can read in netCDF conforming files amongst others. Some of the utilities required to support CF are already supported by CDMS. CDMS will be the basis for the PRISM processing library. The utilities already supported are:

- An extension to udunits to handle non-real-world calendars (i.e. conversion between formatted time, elapsed-since-reference time, and time in components)
- Identify long/lat/vertical/time axes from their attributes.
- Determine the topology of an axis (i.e. does it wrap round).
- Pack/unpack by scale and offset.

Others utilities required are:

- Compress/expand by gathering/scattering.
- Read in `standard_name` table. Is it useful to include equivalencies between `standard_names` and other kinds of common identification e.g. GRIB codes?
- Find a `standard_name` in the table by regexp or pattern matching.
- Look up the unit for a `standard_name`.
- Compute dimensional vertical coordinates using the `formula_terms` attribute.
- Construct the climatology attribute for a climatological time axis.
- Reconstruct the original times from climatological time axes.
- Parse the `cell_methods` attribute.
- Check conformance of a file to the CF standard.

### *Processing library functionalities*

The following is a preliminary prioritized list of the functionalities, which should be available in the processing library. The list of "General Functionalities" was compiled from those in the UKMO PV-WAVE and MPI PINGO libraries. The prioritization is based on input received from the PRISM community.

Further general required/desired features and utilities:

- I/O only when needed (i.e. data not read in until it needs to be operated on, data not written until it needs to be saved).
- Input from a collection of separate files regarded as one logical file, with variables distributed across several files.
- Methods to obtain coordinate information conveniently.

- Change units of data variables.
- Change calendar of time dimension.
- Carry meta-data and data around together when manipulating data variables. Meta-data updated to reflect operations.
- Enquire and modify attributes of data variables and files.
- Insensitivity of operations to ordering of dimensions or sense of coordinates.
- Operator overloading for mathematical operations (so you can write, for example,  $a=b+c$ , where  $a$   $b$   $c$  are objects comprising data and meta-data).

General functionality:	Results:		
	Essential	Important	Optional
Averaging	9		
Arithmetic Operations	8	1	
Information	8	1	
Conversion to new grid	7	2	
Statistical Operations	7	1	1
Extraction of Region and slices	7	1	1
Sampling	6	3	
Gather	6	2	1
Masking	5	3	1
Tests and Confidence Intervals	3	4	2
Vector Operations	3	4	2
Integration and Differentiation	2	5	2
Regression	2	4	3
Mathematical Functions	2	4	3
Filtering and Smoothing	1	6	2
EOF's	2	3	4
Scatter	2	2	5
Classes	1	4	4
Manipulation	1	2	6
Binning Data		1	8

Table 25: Input from Hadley Centre, WP3f-3h, CERFACS, ECMWF, KNMI, MPI, Météo France

## IV.2.5 Visualization

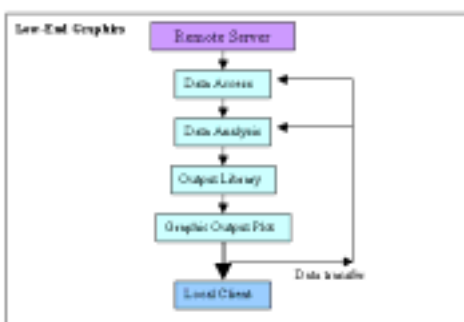


Figure 31: Low-end graphics

Low-end and high-end graphical display systems will be developed. The border between low-end and high-end graphics is defined below.

Definition of low-end Graphics:

*'Low-end graphics refers to the generation of standard plots and animations for quality control, comparison of different runs or models, but they should also be reasonably acceptable for use in presentations and publications.'*

The plan is to provide a system that automatically generates low-end plots on a remote server and allows just the image to be transferred to the local server. A potential solution would be to provide a web interface that allows the user to select the data, assign algorithms and operations, and to choose plot style. The plot is generated on a remote server and the result is displayed in the web browser using a Java applet.

#### Definition of high-end Graphics

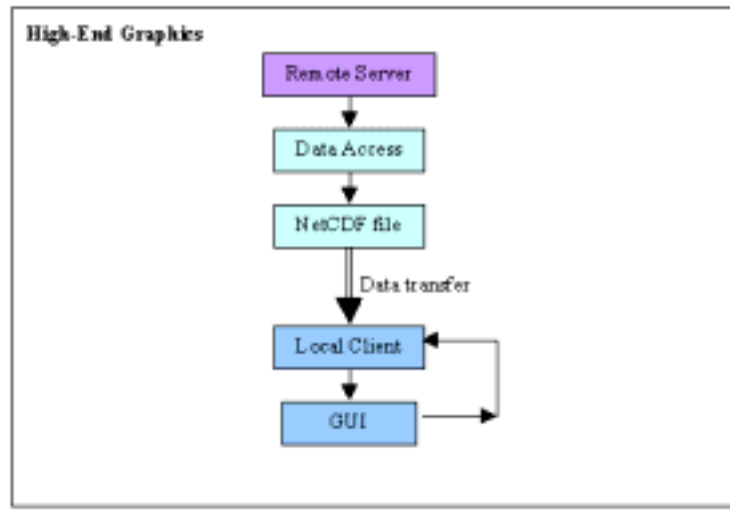


Figure 32: High-end graphics

*'High-end graphics provides facilities to explore the data as well as to generate high quality images for presentations. It includes additional functions such as the ability to have greater flexibility in generating animations, to manipulate 3D images, to rapidly apply new processing operations, and to have close control over all aspects of the displayed images such as the styles, scales and fonts.'*

The need for high performance suggests the need for direct interaction with the data. Therefore the plan is to transfer the data to the local client and carry out any processing and generation of plots here.

In principle each end user should have the choice between several graphic packages depending on local hardware and possibly financial constraints.

All the graphics packages listed in REDOC II.3 have been reviewed using the graphics package questionnaire, drawn up at the 1st visualization meeting in Hamburg, January 2002. The results of this questionnaire can be viewed at

<http://prism.enes.org/WPs/WP4a/Graphics/GraphicPackagesReview-collectionb.html>.

Further investigation regarding the potential of the existing graphics packages is being carried out with focus on those packages that seem to satisfy most of the requirements.

The PRISM community rated the functionalities required in the visualization package, represented by High and Low respectively in the table.

<b>Feature:</b>	<b>Essential</b>	<b>Important</b>	<b>Optional</b>
Ease of Use	High / Low		
Extensibility	High / Low		
Interactive Performance	High / Low		
<b>File Formats:</b>			
NetCDF	High / Low		
HDF		High / Low	
<b>Connectivity:</b>			
Remote	Low	High	
Web Enabled	Low		High
Scripting	High / Low		
Graphical User interface	High	Low	
User Interface Builder			High / Low
<b>Platforms:</b>			
Windows	High		Low
Sun Solaris	High		Low
SGI	High		Low
IBM AIX	High		Low
HP-UX	High		Low
DEC Alpha			High / Low
Linux	High		Low
<b>Limitations:</b>			
Is there a file size limitation?	High / Low		
Multiple File support?	High / Low		
Open source	High / Low		
One kind of scripting language	High / Low		
<b>Types of Grids:</b>			
Regular	High / Low		
Rectilinear	High / Low		
Curvilinear	High / Low		
Unstructured/irregular	High / Low		
Time dependent			High / Low
<b>Plot Types (General):</b>			
1D (lines)	High / Low		
2D (contour)	High / Low		
3D (Surfaces...)	High		Low
3D interactive	High		Low
Animations	High / Low		
Vector Plots	High / Low		
<b>Plots: Methods and Functionalities</b>			
Streamline		High / Low	
Isosurface	High	Low	
Trajectories	High		Low
Particle Trace	High		Low
Volume Rendering	High		Low
Multiple Datasets	High / Low		
Overlays of different grids	High	Low	
Calculating differences between multiple grids		High / Low	
Slices, Subsets	High / Low		
<b>Output File Generations / View mode:</b>			
Video			High / Low
Images	High / Low		
Vector Plot Quality (PS, SVG)	Low	High	
Stereoscopic		High	Low
Cave support			High / Low

Table 26: Required Functionality for the PRISM graphics package

## IV.2.6 Archiving

PRISM will propose an interface for archiving operations and the local PRISM site administrator will be required to customize it with site specific functions to enable interfacing with their archiving system. Connection with and the way in which operations are performed on the file server are site dependent. This could be, for example through direct access to files through the NFS facility, indirect access to files through ftp, scp or other transfer tool, indirect access through a firewall, ...

A file is considered unique if its full file name contains the computer name. E.g. computer.domain:/directory tree/final\_file\_name

A file can be a family of files if special characters are used. For example in the UNIX shell; Year\_200[0-9] for Year\_2000, to Year\_2009

For PRISM we have these requirements:

### **First Priority:**

- Put - to put a file from the supercomputer onto the file server.
- Get - to get a file from a file server and transfer it to a supercomputer.

### **Second Priority:**

- Get - to get a file from a file server remotely and transfer to a local disk.
- Dir - to get a list of files included in a directory on a file server from the file server, supercomputer or local PC.
- Query - to determine if a file exists on a file server from the file server, supercomputer or local PC.

### **Third Priority:**

- Delete - to delete a file on a file server.
- Since there will not be a common PRISM archive (CLIMSTER now not going ahead), it is proposed that each site will be responsible for setting up their own policy for data management.

## IV.2.7 Statement of licensing

Any software developed will be classified as 'Open Source'.



## IV.3 User Interfaces

*The user interface will consist of the PrepIFS GUI, SMS/Xcdp and tools for archiving, visualization and access to experiment results. The PrepIFS GUI will be developed in two versions, as a local system being a basis for development, and as a part of a Web Services infrastructure. No command line interface with remote access capabilities will be provided. Planned new features and adaptations to the PrepIFS Graphical User Interface and the SMS/Xcdp tool for the PRISM project are described.*

---

### IV.3.1 Introduction

The user interface will consist of the PrepIFS GUI, SMS/Xcdp and tools provided by WP4a for archiving, visualization and access to results of the experiment. We refer to these as the UI components in the document.

The use of an existing software and the limitations of time and manpower means that the implementation will concentrate on additional features, adaptations and re-factoring rather than new designs or rewrites.

The user interface will have the functionality as outlined in REDOC II.4 covering the following areas:

- Setup / configuration of an experiment
- Submission and control of execution of an experiment
- Monitoring of an experiment
- Provision of results of an experiment
- Visualization of experiment results
- Access control and security
- On-line Documentation and Help System
- Optional support for model component development

The PrepIFS GUI will be developed in two versions as described in ARCDI III.3 on page 113, "The local system, a basis for development" and as part of a Web Services infrastructure. No attempt will be made to provide a command line interface with remote access capabilities.

This part of the document describes the planned new features and adaptations necessary to the PrepIFS Graphical User Interface and the SMS/Xcdp tool for the PRISM project. One such adaptation is the Administration Interface described in the section with the same name.

### IV.3.2 General Implementation

The user interface components should be able to access resources remotely, i.e. the users do not have to be physically in the same place as where the model is executed or data is located. To achieve this remote access, a web-based system will be implemented, since it allows access to

any resources via the HTTP protocol. It is important that **all** UI components have this capability or the user will still have to have login access to the remote site in the traditional way, defying the objectives of the project.

### IV.3.3 Administration Interface

The task of the administration interface is to enable the flow of information between the model developers who know the specific knowledge necessary to run the model, and the site administrator who has the local knowledge necessary to make the model run at their site. An administrator from each PRISM institution will be responsible for the maintenance and the development of a particular model component of the coupled PRISM system. Furthermore a local administrator at each site will initiate the building of new or updated versions of any PRISM component. This local knowledge will be standardized so that it is accessible by the GUI and will include information such as:

- The path of the local compiler.
- The path of the local linker.
- The path of the local libraries necessary for the PRISM component to run at the local site.
- The path where to build the PRISM components

A service will be developed allowing support for model administration. This functionality will build on the same user interface (PrepIFS) but will configure model builds. Using this interface, new versions of models can be built on remote sites.

### IV.3.4 Adaptations

#### *Input/Output*

The current PrepIFS tool uses a proprietary format for defining the input configuration variables.

Example:

```
TYPE=STRING,NAME=FILEFORMAT,FILEOUT="config.h",FORMAT="ShellFormat",
DEFAULT=general, HTML="/prepifs/an",LABEL="Resolution, general
setup",PRIORITY=true,VIS=false, COMMENT="Most of the resolution parameters
can be defined here",WRITE=false; // Resolution
TYPE=INTEGER,NAME=RESOL,DEFAULT=319, LABEL="Horizontal resolution
(spectral truncation).\n Change this item in the popup menu on the tree",
COMMENT="Change the resolution on the tree...",READONLY=true;
TYPE=SELECTSTRING,NAME=GTYPE,DEFAULT="I_2",RANGE="I_2/_2/_full",
RANGELABEL="Linear reduced gaussian grid/quadratic reduced gaussian grid/full
gaussian grid", LABEL="Grid type";
TYPE=BOOLEANTF,NAME=FPOSINC,DEFAULT=true,LABEL="Use Full-Pos to
interpolate to resolution of simplified model",
FILEOUT="config.h/sms_def",FORMAT="ShellFormat/SMSFormat";
TYPE=INTEGER,NAME=LEVELS,DEFAULT=60, LABEL="Vertical resolution
(number of levels)";
TYPE=BOOLEANTF,NAME=LESUITE,DEFAULT="false",LABEL="Switch for esuite
bit reproducibility check", FILEOUT="config.h",FORMAT="ShellFormat";
TYPE=STRING,NAME=EPSTYPE,DEFAULT="an",LABEL="EPS configuration
part",VIS=false; TYPE=STRING,NAME=EPSMEMBER,DEFAULT="0",LABEL="EPS
Member number",VIS=false;
```

This specification format needs to be standardized by the use of XML as a specification language. In the PRISM project the information present in the existing file format might be split up on multiple files requiring several input operations and a following merge operation.

## *Experiment hierarchy*

The current PrepIFS tool expects a hierarchy of configurations in the following format:

```
Experiment
  !
  !--- ExperimentType X (forecast, analysis etc)
  !
  !--- ExperimentType Y
  !
  !--- ExperimentType Z
!
!
!---NameList A (Archiving, post-processing etc)
!
!---NameList B
!
!---NameList C
  !
  !
  !---Configurable variable I(Archivename, days2run, etc.)
  !
  !---Configurable variable J
  !
  !---Configurable variable K
  !
  !---Configurable variable L
```

This might be unsuitable for the PRISM project and an attempt to decouple the functionality from the levels will be made in order to make it re-assignable to other levels.

## *Authentication module*

A new authentication method will be implemented using password authentication with s/key or other similar scheme. Support for access control will be rudimentary and connected to the user-ID.

## IV.3.5 New features

### *Keyboard Navigation*

The current tool has to support for keyboard navigation through the configuration window. Support for this will be added.

### *Support for New Execution Platforms*

A generalized method will be implemented to describe changes to a configuration, which are necessary to switch between different hardware platforms.

### *Support for remote monitoring*

The Xcdp monitoring tool will be extended to allow monitoring of experiments and execution control.

### *Support for on-line help*

A system that will allow model developers to supply context sensitive help and documentation will be provided.

### *Support for spawning new applications*

The PrepIFS GUI will be extended to allow menu-bar configuration for access to tools for spawning new applications to allow access to results. The applications will be made available by WP 4a.

### *Support for model component development*

A service will be developed allowing model developers to upload software components to the central site and have them compiled and executed with the model (**Optional**).

## V - Software Engineering Process, Coding Rules, and Quality Standards

*By adopting quality assurance techniques during the development process of PRISM components, the generated code will be of greater quality, implying that development and integration times, as well as maintenance effort, can be substantially lowered. We are fully conscious that "The later in the software life cycle an error is detected the higher is the cost of correcting it".*

*Several mechanisms have been proposed targeting quality assurance for the development of PRISM components. Formal coding rules and conventions have been defined in order to improve code portability, code readability and code maintenance in general. Consistent and always up-to-date source code documentation will be ensured by tools like 'ProTex' and 'Doxygen'. All PRISM components, including validation test suites, will be managed by a centralized Code Revision Control mechanism. Test and validation suites will be part of the software development process and will ensure a rapid and early discovery of software problems. Additionally, source code will be peer reviewed.*

---

### V.1 Introduction

This chapter is about issues related to process, conventions, and standards in the design, implementation, and documentation of software that will be developed in the frame of the PRISM project under the consideration of portability, sustained performance and ease of use.

This is an evolving document, part of this document is based on the work done by the *European Cooperation for Space Standardization* extensively covering the whole process of the software development and software life cycle, and on the *"Community Climate System Model, Software Developer's Guide"*

([http://www.ccsm.ucar.edu/csm/working\\_groups/Software/dev\\_guide/dev\\_guide/](http://www.ccsm.ucar.edu/csm/working_groups/Software/dev_guide/dev_guide/)).

The given rules are supposed to provide a guideline for newly written software. Exceptions to all rules are possible, but have to be justified. As a matter of fact software not following the software engineering specification is more likely introducing instability in the system and hence will produce a high amount of technical debugging work. Keep this in mind when arguing that following these rules is too expensive.

### *Target Architecture for the Development of the PRISM System*

The goal of this section is to attempt to propose the most likely target architecture on which the various PRISM components can be integrated into a PRISM system in a first instance. This recommendation is a summary of the conclusions of the document "PRISM REDOC III.2: Review of Current and Future HPC Architectures" (Mangili 2002).

The main HPC facilities available now and most probably to come in the next years are based on clusters of SMP (most of them microprocessor-based, shared-memory inside nodes, distributed-memory between nodes, complemented with fast node interconnects) built either out of RISC or vector CMOS CPUs. Such platforms might have complex memory hierarchies, and often memory bandwidth and latency issues dominate performance. For the PRISM system the first priority must be placed on making the PRISM components run efficiently on these platforms.

Targeting portability on several HPC facilities the PRISM system is likely to be run on both vector and RISC-based CPUs. Ideally, the code would have the flexibility to run efficiently on both. In practice, this can be difficult to achieve, since the choice of optimal data structures, loop ordering, and other significant design decisions may differ depending on whether code is intended for vector or RISC CPUs. This is however extremely important seen that the highest sustained performance for earth science applications will be most probably reached by combining both the powerful CMOS CPUs and the high level application parallelization, as it has been recently successfully demonstrated by the Japanese Earth Simulator. Ideally the PRISM software should run both on the Earth Simulator and ASCI roadmap similar machines.

It is a priority to write flexible code and, whenever possible, to use data structures that are likely to achieve acceptable performance on either architecture. For excellent performance on a given architecture, this can add considerable complexity; the most practical solution for some portions of the model may be the development of two code versions, one scalar and one vector (even if this is discouraged by obvious maintenance problems). In this frame a validation test suite, running regularly on ideally all of the target platforms, testing the different PRISM components and the whole PRISM system is essential.

## V.2 General Software Development Guidelines

In this section we would like to give a general overview of the different steps involved in the software development process. The software life cycle is the sequence in which a project specifies, prototypes, designs, implements, tests, and maintains a piece of software. Explicit recognition of a life cycle encourages development teams to address development issues at the appropriate time; for example, to establish basic software requirements before design or coding begins. We recommend that developers roughly follow the staged delivery model (below) when designing significantly new versions of the full PRISM system and when developing large PRISM components and libraries.

### *The staged delivery model*

The staged delivery model involves the following steps (McConnell 1996):

1. **Software concept:** Collect and itemize the high-level requirements of the system and identify the basic functions that the system must perform.
2. **Software requirements analysis:** Write and review a requirements document, a detailed statement of the scientific and computational requirements for the software. Both scientists and the code development team should review and approve the requirements document.
3. **Software architectural design:** Define a high-level software architecture that outlines the functions, relationships, and interfaces for major components. Write and review an architecture document.
4. **Stage 1, 2... n:** Repeat the following steps creating a potentially releasable product at the end of each stage. Each stage produces a more robust, complete version of the software.
  - **Detailed design:** Create a detailed design document and API specification. Writing code headers instrumented for ProTeX (as described below) can achieve this. Incorporate the interface specification into a detailed design document and review the design.

- **Code construction and unit testing:** Implement the interface, debug and unit test.
- **System testing:** Assemble the complete system; verify that the code satisfies all requirements.
- **Release:** Create a potentially releasable product, including User's Guide and User's Reference Manual. Frequently code produced at intermediate stages software will be used internally.

5. **Code distribution and maintenance:** Official public release of the software, beginning of maintenance phase.

Small, simple pieces of software may not require reviews and separate documents at each stage, but it is still a good idea to prepare at least a design document and review it before implementation.

## V.3 Coding Conventions for the Development of PRISM Components

It is recommended that all new PRISM components follow a coding convention. The goal is to create code with a consistent look and feel so that it is easier to read, understand, port and maintain.

### *Fortran Coding Standard*

This section defines a set of Fortran specifications, rules, and recommendations for the coding of PRISM components. The purpose is to provide a framework that enables users to easily understand or modify code, or to port it to new computational environments. In addition, it is hoped that adherence to these guidelines will facilitate the exchange and incorporation of new packages and parameterizations into PRISM components. Other work that influenced the development of this standard are "Community Climate System Model, Software Developer's Guide", ([http://www.cesm.ucar.edu/csm/working\\_groups/Software/dev\\_guide/dev\\_guide/](http://www.cesm.ucar.edu/csm/working_groups/Software/dev_guide/dev_guide/)), "Report on Column Physics Standards" (<http://nsipp.gsfc.nasa.gov/infra/>) and "European Standards For Writing and Documenting Exchangeable Fortran 90 Code" (<http://nsipp.gsfc.nasa.gov/infra/eurorules.html>).

### **Restriction to the language**

- **Fortran 95 standard** The PRISM coupling software will adhere to the Fortran 95 language standard and not rely on any specific language or vendor extension. The purpose is to enhance portability, and to allow use of the many desirable new features of the language. If a situation arises in which there is good reason to violate this rule and include Fortran code that is not compliant with the f95 standard, an alternate set of f95-compliant code must be provided (tested and validated...). This is normally done through use of a C-pre-processor `#ifdef-#else-#endif` construct.
- **English language** Owing to the international user community, all naming of variables, modules, functions, and subroutines as well as all comments are to be written in English. In case you encounter a situation which you cannot solve in Fortran 95 (e.g. you need to include a library written in C), please make sure the non Fortran code only uses ANSI C with POSIX extensions as described below. This type of code should go into a support library or other specialized libraries.

- Features to be avoided
 

In terms of keeping code up to date and easier to maintain the code should always follow the current standards of Fortran and ANSI C. We would like to restrict the languages use to the elements, which are not obsolete and deleted -- even if they are still available with almost all compilers. Examples for Fortran 95 are:

  - COMMON blocks - use the declaration part of MODULEs instead.
  - EQUIVALENCE - use POINTERS or derived data types instead to form data structures.
  - Assigned and computed GOTOs - use the CASE construct instead.
  - Arithmetic IF statements - use the block IF, ELSE, ELSE IF, END IF or SELECT CASE construct instead.
  - Labeled DO constructs - use unlabeled END DO instead. Due to the structure of the physics routines and the large amount of scientists working on them, we recommend for the ease of communication labeled do constructs of the DO- labeled END DO construct.
  - I/O routines END and ERR - use IOSTAT instead (the use is somehow restricted due compiler implementation dependent error numbering).
  - FORMAT statements: use character parameters or explicit format- specifiers inside the READ or WRITE statement instead.
  - Avoid any unused statement like a labeled CONTINUE not being jumped to.
  - The only sensible use of GOTO is to jump to the error handling section at the end of a routine on detection of an error. The target label should be a CONTINUE statement the label should be 999. However, it is recommended to avoid this practice and use IF, CASE, DO WHILE, EXIT or CYCLE statements or a contained SUBROUTINE instead. If you feel you cannot avoid a GOTO, then add a clear comment to explain what is going on and why you need to use GOTO. Also add a comment to the labeled CONTINUE statement.
  - PAUSE
  - ENTRY statements: a subprogram may only have one entry point.
  - Fixed source form
  - Avoid functions with side effects. Although this is common practice in C, there are good reasons to avoid this. First, the code is easier to understand, if you can rely on the rule that functions don't change their arguments, second, some compilers generate more efficient code for PURE (in Fortran 95 there are the attributes PURE and ELEMENTAL) functions, because they can store the arguments in different places. This is especially important on massive parallel and as well on vector machines.
  - Try to avoid using DATA and BLOCK DATA. Initialisers in Fortran 95 give this functionality.

## *Style Rules*

- **Pre-processor** Where the use of a language pre-processor is required, it will be the C pre-processor (cpp). Cpp is available on any UNIX platform, and many Fortran compilers have the ability to run cpp automatically as part of the compilation process.



- **Free form source** Free-form source will be used. The f95 standard allows up to 132 characters, but a self-imposed limit of 80 should enhance readability and make life easier for those with bad eyesight, who wish to make overheads of source code, or print source files with two columns per page. The world will not come to an end if someone extends a line of code to column 81, but multi-line comments that extend to column 100 for example would be unacceptable.
- **Fortran keywords** Fortran keywords should be written in upper case, the remaining code in lower case.
- **Variables names and declaration**
  - Use meaningful English variable and constant names.
  - Usage of the DIMENSION statement or attribute is not necessary. Declare the shape and size of arrays inside parentheses after the variable name on the declaration statement.
  - The “::” notation is quite useful to show that this program unit declaration part is written in standard Fortran syntax, even if there are no attributes to clarify the declaration section.
  - Declare the length of a character variable using the CHARACTER (len=xxx) syntax - the len specifier is important because it is possible to have several kinds for characters (e.g. Unicode using two bytes per character, or there might be a different kind for Japanese e.g.. NEC).
  - Never use a Fortran keyword as a routine or variable name.
- **Parameter declaration:** Variables used as constants should be declared with attribute `PARAMETER` and used always without copying to local variables. This prevents from using different values for the same constant or changing them accidentally.
- **Program units:** Always name program units and always use the corresponding `END PROGRAM;` `END SUBROUTINE;` `END INTERFACE;` `END MODULE;` etc. construct, again specifying the name of the program unit. This helps finding the end of the current program entity. `RETURN` is obsolete and so not necessary at the end of program units.
- **Loops:** Loops should be structured with the `DO-END DO` construct as opposed to numbered loops.
- **Argument comments:** Input arguments and local variables will be declared 1 per line, with a comment field expressed with a "!" character followed by the comment text all on the same line as the declaration. Multiple comment lines describing a single variable are acceptable when necessary. Variables of a like function may be grouped together on a single line. For example:
 

```
INTEGER :: i,j,k ! Spatial indices
```
- **Continuation lines:** Continuation lines are acceptable on multi-dimensional array declarations that take up many columns. For example:
 

```
REAL(r8), DIMENSION(plond,plev), INTENT(in) :: &
array1, &! array1 is blah blah blah
array2 ! array2 is blah blah blah
```

Note that the ISO/IEC 1539-1:1997 Fortran 95 standard defines a limit of 39 continuation lines.

Code lines, which are continuation lines of assignment statements, must begin to the right of the column of the assignment operator. Similarly, continuation lines of subroutine calls and dummy argument lists of subroutine declarations must have the arguments aligned to the right of the "(" character. Examples of each of these constructs are:

```
a = b + c*d + ... + &
h*g + e*f
```

```
CALL sub76 (x, y, z, w, a, &
b, c, d, e)
```

```
SUBROUTINE sub76 (x, y, z, w, a, &
b, c, d, e)
```

- **Indentation Code** within loops and if-blocks will be indented 3 characters for readability.
- **Spacing conventions**
  - Use blank space, in the horizontal and vertical, to improve readability. In particular try to align related code into columns. For example, instead of:

```
! Initialize Variables
x=1
meaningfulname=3.0_wp
SillyName=2.0_wp
```

write:

```
! Initialize variables
x      = 1
meaningfulname  = 3.0_wp
silly_name     = 2.0_wp
```

- In general use a blank space after a comma (i.e. "a, b, c")
- Do not use tab characters (an extension!) in your code: this will ensure that the code looks as intended when ported.
- **Formatted I/O:** Avoid separating the information to be output from the formatting information on how to output it on I/O statements.
- **Argument list format:** Routines with large argument lists will contain 5 variables per line. This applies both to the calling routine and the dummy argument list in the routine being called. The purpose is to simplify matching up the arguments between caller and callee. In rare instances in which 5 variables will not fit on a single line, a number smaller than 5 may be used. But the per-line number must remain consistent between caller and callee. An example is:

```
CALL linemsvc (u3(i1,1,1,j,n3m1), v3(i1,1,1,j,n3m1), &
              t3(i1,1,1,j,n3m1),    q3(i1,1,1,j,n3m1), &
              qfcst(i1,1,m,j),      xxx)
SUBROUTINE linemsvc ( u,      v, &
                    t,      q, &
```

qfcst, xxx)

- **Array arguments:** Do not implicitly change the shape of an array when passing it into a subroutine. Although actually forbidden in the FORTRAN77 standard it was very common practice to pass n dimensional arrays into a subroutine where they would, say, be treated as a one dimensional array. Though officially banned in Fortran 95, this practice is still possible with external routines for which no interface block is supplied. The danger of this method is that it makes certain assumptions about how the data is stored.
- **Commenting style:** Short comments may be included on the same line as executable code using the "!" character followed by the description. More in-depth comments should be written in the form:

```
!  
! Describe what is going on  
!
```

Key features of this style are 1) it starts with a "!" in column 1; 2) The text starts in column 3; and 3) the text is offset above and below by a blank comment line. The blank comments could just as well be completely blank lines (i.e. no "!") if the developer prefers.

- **Operators:** Use of the operators <, >, <=, >=, ==, /= is recommended instead of their deprecated counterparts, lt., .gt., .le., .ge., .eq., and .ne. The motivation is readability. In general use the notation: <Blank><Operator><Blank>
- **File format:** Embedding multiple routines within a single file and/or module is allowed, encouraged in fact, if any of three conditions hold. First, if routine A and only routine A is routine B, then the two routines may be included in the same file. This construct has the advantage that inlining B into A is often much easier for compilers if both A and B are in the same file. Practical experience with many compilers has shown that inlining when A and B are in different files often is too complicated for most people to consider worthwhile investigating.

The second condition in which it is desirable to put multiple routines in a single file is when they are "CONTAIN"ed in a module for the purpose of providing an implicit interface block. This type of construct is strongly encouraged, as it allows the compiler to perform argument consistency checking across routine boundaries. An example is:

```
file 1:  
  SUBROUTINE driver  
  USE mod1  
  REAL :: X, Y  
  ...  
  CALL sub1(X,Y)  
  CALL sub2(Y)  
  ...  
  END SUBROUTINE driver
```

```
file 2:  
  MODULE mod1  
  
  PRIVATE
```

```

PUBLIC sub1, sub2

CONTAINS

SUBROUTINE sub1(A,B)
...
END SUBROUTINE sub1

SUBROUTINE sub2(A)
...
END SUBROUTINE sub2
END MODULE mod1

```

The compiler automatically checks the number, type, and dimensions of the arguments passed to sub1 and sub2.

The final reason to store multiple routines and their data in a single module is that the scope of the data defined in the module can be limited to only the routines which are also in the module. This is accomplished with the "private" clause.

If none of the above conditions hold, it is not acceptable to simply glue together a bunch of functions or subroutines in a single file.

- **Module names:** Modules MUST be named the same as the file in which they reside. The reason to enforce this as a hard rule has to do with the fact that dependency rules used by "make" programs are based on file names. For example, if routine A "USE"s module B, then "make" must be told of the dependency relation which requires B to be compiled before A. If one can assume that module B resides in file B.o, building a tool to generate this dependency rule (e.g. A.o: B.o) is quite simple. Put another way, it is difficult (to say nothing of CPU-intensive) to search an entire source tree to find the file in which module B resides for each routine or module which "USE"s B.

Note that by implication multiple modules are not allowed in a single file.

The use of common blocks is deprecated in Fortran 95 and their continued use in the PRISM component is strongly discouraged. Modules are a better way to declare static data. Among the advantages of modules is the ability to freely mix data of various types, and to limit access to contained variables through use of the ONLY and PRIVATE clauses.

- **Array syntax:** Array notation should be used whenever possible. This should help optimization regardless what machine architecture is used (at least in theory) and will reduce the number of lines of code required. To improve readability the array shape should be shown in brackets, e.g.:

```

onedarraya(:) = onedarrayb(:) + onedarrayc(:)
twodarray(:, :) = scalar * anothertwodarray(:, :)

```

When accessing sections of arrays, for example in finite difference equations, do so by using the triplet notation on the full array, e.g.:

```

twodarray(:,2:len2) = scalar &
                    * (anothertwodarray(:,1:len2-1) &
                    - anothertwodarray(:,2:len2))

```

Note: In order to improve readability of long, complicated loops, explicitly indexed loops should be preferred. In general when using this syntax, the order of the loops indices should reflect the following scheme: (best usage of data locality).

```
DO k = 1, nk
  DO j = 1, nj
    DO i = 1, ni
      f(i, j, k) = ...
    END DO
  END DO
END DO
```

## Content rules

- **Implicit None:** All subroutines and functions will include an "implicit none" statement. Thus all variables must be explicitly typed. It also allows the compiler to detect typographical errors in variable names. For **MODULES**, one **IMPLICIT NONE** statement in the modules definition section is sufficient.
- **Prologues:** Each function, subroutine, or module will include a prologue instrumented for use with the ProTeX auto-documentation script (<http://dao.gsfc.nasa.gov/software/protex>). The purpose is to describe what the code does, possibly referring to external documentation. The prologue formats for functions and subroutines, modules, and header files are shown. In addition to the keywords in these templates, ProTeX also recognizes the following:

```
!BUGS:
!SEE ALSO:
!SYSTEM ROUTINES:
!FILES USED:
!REMARKS:
!TO DO:
!CALLING SEQUENCE:
!CALLED FROM:
!LOCAL VARIABLES:
```

These keywords may be used at the developer's discretion. We would like to recommend to use the REMARKS part to add an responsible author, who can be contacted in case of problems.

## Prologue for functions and subroutines

If the function or subroutine is included in a module, the keyword **!ROUTINE** should be used instead of **!ROUTINE**

```
!-----
! BOP
!
! !ROUTINE:   <Function name> (!ROUTINE if the function is in a module)
!
! !!INTERFACE:
           function <name> (<arguments>)
```

```

! !USES:
    use <module>

! !RETURN VALUE:
    implicit none
    <type> :: <name>                ! <Return value description>

! !PARAMETERS:
    <type, intent> :: <parameter>    ! <Parameter description>

! !DESCRIPTION:
! <Describe the function of the routine and algorithm(s) used in
! the routine. Include any applicable external references.>
!
! !REVISION HISTORY:
! YY.MM.DD <Name> <Description of activity>
!
! EOP
!-----
! $Id: code_conv_cam.tex,v 1.3 2001/06/19 21:44:14 kauff Exp $
! $Author: kauff $
!-----

```

## Prologue for a module

```

!-----
! BOP
!
! !MODULE:      <Module name>
!
! !USES:
    use <module>

! !PUBLIC TYPES:
    implicit none
    [save]

    <type declaration>

! !PUBLIC MEMBER FUNCTIONS:
!     <function>                ! Description
!
! !PUBLIC DATA MEMBERS:
    <type> :: <variable>        ! Variable description

! !DESCRIPTION:
! <Describe the function of the module.>

```

```

!
! !REVISION HISTORY:
! YY.MM.DD <Name> <Description of activity>
!
! EOP
!-----
! $Id: code_conv_cam.tex,v 1.3 2001/06/19 21:44:14 kauff Exp $
! $Author: kauff $
!-----

```

### Prologue for a header file

```

!-----
! BOP
!
! !INCLUDE:   <Header file name>
!
! !DEFINED PARAMETERS:
!             <type> :: <parameter>      ! Parameter description
!
! !DESCRIPTION:
! <Describe the contents of the header file.>
!
! !REVISION HISTORY:
! YY.MM.DD <Name> <Description of activity>
!
! EOP
!-----
! $Id: code_conv_cam.tex,v 1.3 2001/06/19 21:44:14 kauff Exp $
! $Author: kauff $
!-----

```

- **I/O error conditions:** I/O statements which need to check an error condition will use the "iostat=<integer variable>" construct instead of the outmoded `end=` and `err=`. Note that a 0 value means success, a positive value means an error has occurred, and a negative value means the end of record or end of file was encountered.
- **Intent:** All dummy arguments must include the `INTENT` clause in their declaration. This is extremely valuable to someone reading the code, and can be checked by compilers. An example is:

```

SUBROUTINE sub1 (x, y, z)
IMPLICIT NONE
REAL(r8), INTENT(in) :: x
REAL(r8), INTENT(out) :: y
REAL(r8), INTENT(inout) :: z

```

```

y = x
z = z + x

END SUBROUTINE sub1

```

## Package coding rules

The term "package" in the following rules refers to a routine or group of routines, which takes a well-defined set of input and produces a well-defined set of output. A package can be large, such as a dynamics package, which computes large-scale advection for a single time-step. It can also be relatively small, such as a parameterization to compute the effects of gravity wave drag.

- **Self-containment:** A package should refer only to its own modules and subprograms and to those intrinsic functions included in the Fortran95 standard. This is crucial to attaining plug-compatibility. An exception to the rule might occur when a given computation needs to be done in a consistent manner throughout the model. Thus for example a package, which requires saturation vapour pressure, would be allowed to call a generic routine used elsewhere in the main model code to compute this quantity. When exceptions to the above rule apply, (i.e. routines are required by a package which are not f95 intrinsics or part of the package itself) the required routines that violate the rule must be specified within the package.
- **Single entry point:** A package shall provide separate setup and running procedures, each with a single entry point. All initialization of time invariant data must be done in the setup procedure and these data must not be altered by the running procedure. This distinction is important when the code is being run in a multitasked environment. For example, constructs of the following form will not work when they are multitasked:

```

SUBROUTINE sub
  LOGICAL first/.true./
  IF (first) THEN
    first = .false.
    <set time-invariant values>
  END IF

```

- **Interface blocks:** Explicit interface blocks are required between routines if optional or keyword arguments are to be used. They also allow the compiler to check that the type, shape and number of arguments specified in the CALL are the same as those specified in the subprogram itself. Fortran 95 compilers can automatically provide explicit interface blocks for routines contained in a module.
- **Communication:** All communication with the package will be through the argument list or namelist input. The point behind this rule is that packages should not have to know details of the surrounding model data structures, or the names of variables outside of the package. A notable exception to this rule is model resolution parameters. The reason for the exception is to allow compile-time array sizing inside the package. This is often important for efficiency.
- **Package attribute:** Modules variables and routines should be encapsulated by using the `PRIVATE` attribute. What shall be used outside the module can be declared `PUBLIC` instead. Use `USE` with the `ONLY` attribute to specify which of the variables, type definitions etc. defined in a module are to be made available to the using routine. Of course you don't need to add the `ONLY` attribute if you include the complete module or almost all of its public declarations.



- **Precision:** Parameterizations should not rely on vendor-supplied flags to supply a default floating point precision or integer size. The f95 `KIND` feature should be used instead.

In order to improve portability between 32 and 64 bit platforms, it is necessary to make use of kinds by providing a specific module declaring the "kind definitions" to obtain the required numerical precision and range as well as size of `INTEGER`. It should be noted that constants need to have attached a `_kindvalue` to have the according size.

Example, with `dp` as a real kind with 12 significant digits (usually double precision, 8 byte), and `wp` a working precision:

```
USE my_kind, ONLY: dp, wp
IMPLICIT NONE

! Declaration of a constant
REAL(dp), PARAMETER :: pi = 3.14159265358979323846_dp
! Declaration of a 3d field
REAL(wp), POINTER :: geopotential(:, :, :)

! Declaration of a local variable
REAL(wp) :: a
...
a = 4.0_wp
...
```

Possible kinds in a proposed `my_kind`:

Note:

The bit sizes given are not mandatory. The kind value is selected regarding a given precision, which results on current systems in this bit sizes. This may change in future.

A portable way to build a module providing several kinds is given in the following example:

```
MODULE my_kind
  IMPLICIT NONE

  !Number model from which the
  !      SELECTED_*_KIND are requested:
  !
  !
  !           4 byte REAL   8 byte REAL
  !      CRAY:   - precision =   13
  !              exponent = 2465
  !      IEEE:   precision =6   precision =   15
  !              exponent= 37   exponent =   307
  !
  !Most likely this are the only possible models.
```

Kind variable	Assumed number of bits
<b>real variables</b>	
sp	32
dp	64
<b>integer variables</b>	
i4	32
i8	64
<b>Integer to contain a C pointer</b>	
cp	bit size of a C pointer

```

!Floating-point section

INTEGER, PARAMETER :: sp = SELECTED_REAL_KIND(6,37)
INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND(12,307)

INTEGER, PARAMETER :: wp = dp    ! working precision

! Integer section

INTEGER, PARAMETER :: i4 = SELECTED_INT_KIND(9)
INTEGER, PARAMETER :: i8 = SELECTED_INT_KIND(14)

! Pointer section, the cp type should provide an integer
! with a size for transporting a C pointer through Fortran

#ifdef CP4
                                INTEGER, PARAMETER :: cp = i4
#else
                                INTEGER, PARAMETER :: cp = i8
#endif

END MODULE my_kind

```

Thus, any variable declared `real(dp)` will be of sufficient size to maintain 12 decimal digits in their mantissa. Likewise, integer variables declared `integer(i8)` will be able to represent an integer of at least 13 decimal digits.

- **Bounds checking:** All PRISM software and PRISM components must be able to run when a compile-time and/or run-time array bounds checking option is enabled. Thus, constructs of the following form are disallowed:

```
REAL(r8) :: arr(1)
```

where "arr" is an input argument into which the user wishes to index beyond 1. Use of the (\*) construct in array dimensioning to circumvent this problem is forbidden because it effectively disables array bounds checking.

- **Error conditions:** When an error condition occurs inside a package, a message describing what went wrong will be printed. The name of the routine in which the error occurred must be included. It is acceptable to terminate execution within a package, but the developer may instead wish to return an error flag through the argument list. If the user wishes to terminate execution within the package, a generic PRISM Coupling Software termination routine "endrun" should be called instead of issuing a Fortran "stop". Otherwise a message-passing version of the model could hang. Note that this is an exception to the package-coding rule that "A package should refer only to its own modules and subprograms and to those intrinsic functions included in the Fortran 95 standard".
- **Inter-procedural code analysis:** Use of a tool to diagnose problems such as array size mismatches, type mismatches, variables which are defined but not used, etc. is strongly encouraged. Flint is one such tool that has proved valuable in this regard. It is not a strict rule that all PRISM codes and packages must be "flint-free", but the developer must be able to provide adequate explanation for why a given coding construct should be retained

even though it elicits a complaint from flint. If too many complaints are issued, the diagnostic value of the tool diminishes toward zero.

- **Memory management:** The use of dynamic memory allocation is not discouraged because we realize that there are many situations in which run-time array sizing is desirable. However, this type of memory allocation can cause performance problems on some machines, and some debuggers get confused when trying to diagnose the contents of such variables. Therefore, dynamic memory allocation is allowed only "when necessary". The ability to run a code at a different spatial resolution without recompiling is not considered to be an adequate reason to use dynamically allocated arrays.

The preferable mechanism for dynamic memory allocation is automatic arrays, as opposed to ALLOCATABLE or POINTER arrays for which memory must be explicitly allocated and de-allocated. An example of an automatic array is:

```
SUBROUTINE sub(n)
  REAL :: a(n)
  ...
END SUBROUTINE sub
```

The same routine using an allocatable array would look like:

```
SUBROUTINE sub(n)
  REAL, ALLOCATABLE :: a(:)

  ALLOCATE(a(n))
  ...
  DEALLOCATE(a)
  ...
END SUBROUTINE sub
```

- **Constants and magic numbers:** Magic numbers should be avoided. Physical constants (e.g. pi, gas constants) must NEVER be hardwired into the executable portion of a code. Instead, a mnemonically named variable or parameter should be set to the appropriate value, probably in the setup routine for the package. We realize that many parameterizations rely on empirically derived constants or fudge factors, which are not easy to name. In these cases it is not forbidden to leave such factors coded as magic numbers buried in executable code, but comments should be included referring to the source of the empirical formula.

Hard-coded numbers should never be passed through argument lists. One good reason for this rule is that a compiler flag, which defines a default precision for constants, cannot be guaranteed. Fortran95 allows specification of the precision of constants through the "\_" compile-time operator (e.g. 3.14\_dp or 365\_i8). So if you insist on passing a constant through an argument list, you must also include a precision specification in the calling routine. If this is not done, a called routine that declares the resulting dummy argument as, say, real(dp) or 8 bytes, will produce erroneous results if the default floating point precision is 4 byte.

The PRISM software will distribute some physical variables/constants, which are shared between several PRISM model components.

## Interface to other languages

- **Fortran / C Interface:** Interfaces between Fortran and C (C routines called by FORTRAN) should be based on the cfortran package (<http://wwwinfo.cern.ch/asd/cernlib/cfortran.html>).

## *FORTRAN code parallelization issues*

- **Parallelization paradigms:** Parallelization should be done with well-documented and commonly accepted paradigms like MPI, OpenMP and Pthreads.
- **Thread-safe:** The coupling software and other modules callable by any component of the PRISM system should be thread-safe to allow for a flexible usage of hierarchical programming models.
- **Scalability:** Each module of the coupling software and the components to be coupled should follow coding strategies for parallelization, which allow for runs on high processor counts. Parallelized model components should be programmed flexible with regard to the number of processors, which can be used for parallel runs. If necessary hierarchical programming techniques should be used to achieve that goal.
- **Data locality:** Regarding to parallelization the coding techniques should be concerned about memory hierarchies of modern computer architectures. A key to parallel efficiency is the locality of the memory references. On a SMP-like architecture one should constrain memory references within a compute node as much as possible rather than excessively accessing remote memory location since networks between compute nodes of these SMP-like architectures are slower than intra-node memory paths in general. A similar point of view is valid for intra-node memory hierarchies like caches.
- **Data and process placement:** The environment of a parallel run (scripts, configuration files, etc.) and the parallelized components of a PRISM coupled system should allow for a flexible placement or mapping of the various tasks of a coupled simulation onto the computer resources like compute nodes, file systems, etc.

## *C Coding standard*

This section defines a set of C specifications, rules, and recommendations for the coding of PRISM components. The purpose is to provide a framework that enables users to easily understand or modify the code, or to port it to new computational environments. In addition, it is hoped that adherence to these guidelines will facilitate the exchange and incorporation of new packages and parameterizations into the model. We would like to adhere to the "GNU Coding Standard" Chap. 5 "Making the Best Use of C",

([http://www.gnu.org/prep/standards\\_toc.html](http://www.gnu.org/prep/standards_toc.html)).

Some of the obvious rules already given in the section before and an extension to the GNU Coding Standard will be repeated here.

## **Restriction to the language**

- **C ANSI standard** The PRISM coupling software will adhere to the ANSI C language standard with POSIX extensions (ISO/IEC 9899:1990 compliant) and not rely on any specific language or vendor extension. The purpose is to enhance portability across different

platforms. If a situation arises in which there is good reason to violate this rule and include C code that is not compliant with the ANSI C standard, an alternate set of ANSI C compliant code must be provided (tested and validated...). This can be achieved by use of the C-pre-processor.

- **English language** Owing to the international user community, all naming of variables, modules, functions, and subroutines as well as all comments are to be written in English.

## Content rules

C code should include comment blocks detailing code behaviour according to the rules of the open-source auto-documentation system doxygen (<http://www.doxygen.org/>) whose user manual is available online at <http://www.stack.nl/~dimitri/doxygen/manual.html>.

Before each function or class declaration, a comment block should be added exactly detailing the calling interface and the optional return values. The purpose is to exactly describe what the code does, possibly referring to external documentation.

Doxygen comment blocks for C source are in the following format:

```
/*! \fn test
    ... text ...
    \param c1 an integer
    \param f2 a float
    \return a character pointer
*/
char *test(int c1; float f2) {
...
}
```

and must precede the declaration, rather than the definition of the code unit to be documented.

Special commands are used to indicate the type of program unit that is being documented, they are:

- `\class` to document a C++ class
- `\struct` to document a C-struct
- `\union` to document a union
- `\enum` to document an enumeration type
- `\fn` to document a function
- `\var` to document a variable or typedef or enum value
- `\def` to document a #define
- `\file` to document a file
- `\namespace` to document a namespace

Many more special commands are used for text formatting. See the doxygen manual for a description of those and many detailed examples.

## Interface to other languages

- **Fortran / C Interface** Interfaces between Fortran and C (Fortran routines called by C) should be based on the cfortran package (<http://wwwinfo.cern.ch/asd/cernlib/cfortran.html>).

## C Code parallelization issues

Please refer to "FORTRAN Code Parallelization Issues" above.

## V.4 Component and Unit Testing

Quite complex and interdependent software such as PRISM components requires extensive testing in order to prevent system defects and to provide stable, reliable, and solid software to work with.

Layered testing has shown to be the most effective in catching software defects. Layered testing refers to testing on different levels, both testing individual subroutines as well as more complex systems. There may be several layers of simple to more complex systems tested as well. Testing the individual component models stand-alone is an example of a system less complex than the entire PRISM system.

Unit testing is the first layer, meaning testing individual subroutines or modules. Unit testing alone will not catch defects that are dependent on relationships between different modules, but testing the entire system sometimes will not catch errors within an individual module. That is why using both extremes is useful in catching model defects. Section V.5 covers testing for the entire PRISM system, this section goes over testing of individual model components and unit testing of subroutines and modules within those components.

Since the PRISM system and PRISM components take substantial computer resources to run, catching errors early can also cut computing costs significantly. In addition to that as pointed out by McConnell (1993) development time decreases dramatically when formal quality assurance methods including code reviews are implemented as described in section 7.

### *Designing Good Component Tests*

Each PRISM component needs to develop and maintain its own suite of testing for that given component. Each component development team should provide a written analysis of the testing procedure and testing plan required by the components they are developing. An automated procedure has to be provided to run a suite of standard tests, the result of this procedure is either a **'PASSED'** or **'FAILED'** result, this will be useful to ensure the models work and continue to work as needed. This is especially useful for making sure PRISM components continue to work on multiple platforms.

In order to design a comprehensive testing plan we want to take advantage of the following types of tests:

- **Unit tests:** Testing done on a single subroutine or module.
- **Functionalities:** Testing for a given functional group of subroutines or modules, for example, testing model dynamics alone without the model physics.

- **Component tests:** Testing done on the whole component.

## **Unit tests**

Unit tests are a good way to flush out certain types of defects. Since unit tests only run on one subroutine they are easier to use, faster to build and run, allow more comprehensive testing on a wider range of input data, help document how to use and check for valid answers, and allows faster testing of individual pieces. By building and maintaining unit tests the same tests can be run and used by other developers as part of a more comprehensive testing package. Without maintaining unit tests developers often do less testing than required (since component tests are so much harder to do) or they have to "hack" together their own unit tests for each change. By maintaining unit tests we allow others to leverage off previous work and provide a format to quickly do extensive checking.

Good unit tests will do the following:

1. Applicable requirements are checked.
2. Exercise every line of code.
3. Check that the full range of possible input data works. (i.e. if Temperature is input check that values near both the minimum and maximum possible values work)
4. Boundary analysis. Logical statements that refer to threshold states are checked to ensure they are correct.
5. Check for bad input data.
6. Test for scientific validity.

By analysing the code to be tested different test cases can be designed to ensure that all logical statements are exercised in the unit tests. Similarly input can be designed to test logical threshold states (boundary analysis). Testing scientific validity is of course the most difficult. But, sometimes testing states where the answer is known analytically can be useful. And ensuring (or measuring) the degree to which energy, heat, or mass is conserved for conservative processes can also often be done. These types of tests may also be applied for more complex functional and component tests as well.

## **Functionalities**

Functional tests take a given sub-set of the component and test this set for a particular functionality. Scientific functional tests are common. Important functional tests should be maintained in the source code revision system as separate modules that include the directories maintained for the main component and be distributed with the PRISM component.

## **Component tests**

Component tests need to ensure that the given PRISM component compiles, builds, and runs and that it passes important model requirements. For example, restart capabilities.

## PRISM testing requirements and implementation details

**Unit and Functional tests** should meet the following minimum requirements:

1. Maintained in CVS either with the rest of the models source code or as a separate module that can be used. Module and/or directory name should be easily identifiable such as unit test.
2. Have documentation on how to use it.
3. Have a Makefile associated with it. It may be useful to leverage off the main Makefile so that the compiler options are the same and so that platform dependencies don't have to be maintained twice.
4. Check error conditions so that an error will print out problems.
5. Interactive tests should be avoided, unless a completely automatic test can be set-up using the same mechanism (i.e. redirecting the input).
6. In general unit tests should be run with as many compiler debug options on as possible (bounds checking, signal trapping etc.) as well as using optimized options, making sure the results are comparable.

**Component tests** should meet the following minimum requirements:

1. Ensure that the given model will compile, build and run on at least one production platform.
2. Ensure that the given model will work with the PRISM system on at least one production platform.

## V.5 System Testing and Validation

Regular system testing and validation of the whole PRISM system is required to ensure that components quality and integrity is maintained throughout the development process. This section establishes the system testing standards and the procedures that will be used to verify the standards have been met. It is assumed that PRISM component development teams have 'unit-tested' their component prior to making it available for system testing. See section V.4 for more information on testing of individual components and unit testing of individual subroutine and modules within components.

There are two general categories of model evaluations: *frequent short* test runs and *less frequent long* validation integrations.

**Model testing** refers to short (few days to one month) model runs designed to verify that the underlying mechanics and performance of the coupled models continues to meet specifications. This includes verifying that the model actually starts up and runs, benchmarking model performance and relative speed/cost of each model component as well as checking features like restarting capabilities. These tests are done on each of the target platforms. Model testing does not address whether the model answer is correct, it merely verifies that it mechanically operates as specified.

**Model validation** involves longer (at least 1 year) integrations to ensure that the model results are in acceptable agreement with both previous model climate statistics and observed characteristics of the real climate system. Model validation occurs with each minor PRISM system version (i.e.



PRISM\_2.1, PRISM\_2.2) or on request (PRISM scientists or working groups). Once requested, model validation is only carried out after PRISM scientists have been consulted and the 'Model testing' phase has been successfully completed. The model validation results are documented on the prism web pages <http://prism.enes.org>.

**Port validation** is defined as verification that the differences between two otherwise identical configuration simulations obtained on different machines or using different environments are caused by machine round-off errors only.

### *Testing procedures for the PRISM system*

Formal testing of the PRISM system is required for each tagged version of the coupling software and the component models. The PRISM quality assurance leader is responsible for ensuring that these tests are run, either by personally doing it or having them run by a qualified person. If a model component is identified as having a problem, the component development team is expected to make resolving that problem their highest priority. The results of the testing and benchmarking will be included in the tagged model to document the run characteristics of the model. The actual testing and analysis scripts will be part of the PRISM software code repository to encourage use by outside users.

#### **Development testing steps**

1. *Successful build:* PRISM coupling software and the component models shall compile on each of the target platforms with no changes to the scripts, codes or datasets.
2. *Successful startup:* PRISM will start from an initial state and run for a specified period depending on the component models and experiment setup.
3. *Performance benchmarking:* The total CPU time, memory usage, output volume, disk space use and wall clock time for a run with the specified length will be recorded. The relative cost of each component will also be recorded.
4. *Test report:* The results of all steps above are to be documented in a test report with emphasis on results, comparisons to the previous test and recommendations for improvements. Any faults or defects observed shall be noted and must be brought to the attention of the responsible for that component and the software engineering manager.

### *Model validation procedures for the PRISM system*

Model validation occurs with a new model version or at the request of the PRISM scientists and working groups. Before starting a validation run, the PRISM Quality Assurance Leader will consult with the PRISM scientists to design the validation experiment.

#### **Pre-validation steps:**

1. *Tests run successfully* The validation will successfully complete the testing steps outlined above.

2. *Scientist sign-on* The PRISM scientists must agree to make themselves available to informally analyse the results of the run during the run and formally review the results within one week of the completion of the run.

**Validation steps:**

1. *Comparison with previous model runs* the signed-on scientist accepts Results.
2. *Comparison with observed climate* Result agrees with observed climate

### *Port validation of the PRISM system*

*Port validation* is defined as verification that the differences between two otherwise identical model simulations obtained on different machines or using different environments are caused by machine round-off errors only. Round-off errors can be caused by using two machines with different internal floating point representation, or by using a different number of processing elements on the same machine which may cause a known re-ordering of some calculations, or by using different compiler versions or options (on a single machine or different machines) which parse internal computations differently.

As established in Rosinski and Williamson (1997), three conditions of model solution behaviour must be fulfilled to successfully validate a port of atmospheric general circulation models:>

1. During the first few time-steps, differences between the original and ported solutions should be within one to two orders of magnitude of machine rounding;
2. During the first few days, growth of the difference between the original and ported solutions should not exceed the growth of an initial perturbation introduced into the lowest-order bits of the original solution;

The statistics of a long simulation must be representative of the climate of the model as produced by the original code.

The extent to which these conditions apply to models other than an atmospheric model has not yet been established. Also, note that the third condition is not the focus of this section.

## V.6 Code Maintenance Issues

This section summarizes code maintenance issues and challenges related to the PRISM project and gives an brief overview on how this problem will be addressed. The software maintenance process is a tedious and often costly activity, it is a fundamental part of the software engineering process that cannot be just avoided or simply ignored. The software engineering literature is plenty of examples of died projects simply because the maintenance costs just exploded. Still several actions and measures can be undertaken in order to minimize the effort of the maintenance process. In the PRISM project we tried to find out the right compromise of software engineering rules and conventions being conscious and considering that most of the PRISM components are software packages already developed with their own software life cycle and consequently their own software maintenance process already in place since years.

It is recommended that all PRISM components follow some general guidelines for basic code maintenance as a code revision control and a code configuration, building. The goal is to provide a single code repository with a consistent and unique revision control scheme, as well as to provide an easier port, installation, building and validation on different architectures.

## *Software revision control*

It is recommended that all PRISM development teams develop code with the help of a robust software configuration management (SCM) tool. The goal is to establish, document, control, and track the evolution of source code and related documentation throughout the software life cycle.

As part of the PRISM system this software configuration management tool is CVS.

### **What is CVS?**

CVS stands for Concurrent Versions System and is a version control system. Using it, one can record the history of his source files.

For example, bugs sometimes creep in when software is modified and one might not detect the bug until a long time after the modification was made. With CVS, one can easily retrieve old versions to see exactly which change caused the bug. This can sometimes be a big help.

CVS stores all the versions of a file in a single file in a clever way that only stores the differences between versions.

CVS also helps developers when they are part of a group of people working on the same project. It is all too easy to overwrite each other's changes unless one is extremely careful. Some editors, like GNU Emacs, try to make sure that two people never modify the same file at the same time. Unfortunately, if someone is using another editor, that safeguard will not work. CVS solves this problem by insulating the different developers from one another. Every developer works in his own directory and CVS merges the work when each developer is done.

### **What is CVS not?**

- **CVS is not a build system.** Though the structure of a repository and modules file interact with a build system (e.g. `Makefile's), they are essentially independent. CVS does not dictate in any way in which anything is built. It merely stores files for retrieval in a previously planned tree structure.
- **CVS is not a substitute for management.** For instance, the dates of schedules, release dates or branch points are up to people working on a certain project. If certain milestones are not kept, CVS can't help.
- **CVS is not a substitute for communication.** A bug fix of a source module has to be reviewed before releasing it or checking it into the official source tree.
- **CVS does not have change control** Change control refers to a number of things. First of all it can mean bug tracking, that is being able to keep a database of reported bugs and the status of each one (is it fixed? in what release? has the bug submitter agreed that it is fixed?). For interfacing CVS to an external bug tracking system, see the `rcsinfo' and `verifymsg' files (see section C. Reference manual for Administrative files). Another aspect of change control is keeping track of the fact that changes to several files were in fact changed together as one logical change. If one has checked in several files in a single

CVS commit operation, CVS then forgets that those files were checked in together, and the fact that they have the same log message is the only thing tying them together. Keeping a GNU style `ChangeLog` can help somewhat. Yet another aspect of change control, in some systems, is the ability to keep track of the status of each change. Some changes have been written by a developer, others have been reviewed by a second developer, and so on. Generally, the way to do this with CVS is to generate a diff (using CVS diff or diff) and email it to someone who can then apply it using the patch utility. This is very flexible, but depends on mechanisms outside CVS to make sure nothing falls through the cracks.

- **CVS is not an automated testing program** It should be possible to enforce mandatory use of a test suite.

## CVS Features

As mentioned CVS stores the changes of a source in a repository in a compact way. That repository can be located on a remote server. The access to the remote server is controlled by login id, password, and file permissions and by the so-called CVS tags. For remote login it is even possible to define the remote shell. This can be even the secure shell which gives the administrator of the repository even more access control ( Is the accessing host trusted? Did the user provide a public key? etc.).

CVS allows to define branches of a source tree and individual access rights for that branches. Complete individual branches can be accessed via tags that allow to check out sources, which were logically built as one single unit, according to the changes that have been applied. Different branches can be merged.

CVS allows to check in and check out sources with automatic provision of a release number and history of changes applied.

One can compare releases of individual source files or complete branches.

CVS has a mechanism to ask the developer for comments concerning the changes applied before checking in. These comments are kept alongside with the changes.

CVS has a locking mechanism, which prevents race conditions during a source check in into the same branch. A developer can query the status of file and get information about who else has checked out a file for editing.

Last but not least CVS provides keyword substitution, i.e.. the keyword '\$Header\$' is substituted by the full path name of the RCS file, the revision number, the date (UTC), the author, the state, and the locker (if locked).

For more information we suggest to refer to the reference manual. Moreover, we would like to mention that a GUI clients exist:

Tk based: tkCVS (<http://www.twobarleycorns.net/tkcv.html>)

Java based: jCVS (<http://www.jcvs.org/>)

## CVS Repository access

The PRISM CVS Repository will be installed and accessible under <http://prism.enes.org/>.

## Recommendations for code developers

We strongly recommend the use of the keyword '\$Header\$'. CVS keywords are often only used within source code comments, we encourage developers to additionally define a static string variable storing the expanded keyword information. The advantage consists in the possibility of tracking module revisions within the binary, by using tools like 'whatis' or 'rcsinfo'. This greatly helps with bug analysis:

After checking out a code for editing the developer may therefore insert these lines in its program. In either C:

```
static char PRISM_CVSID[]="$Header$"
```

or Fortran 95:

```
CHARACTER(len=80), PARAMETER:: PRISM_CVSID='$Header$'
```

## Code Maintenance Process

The maintenance process is related to activities and tasks of the software maintainer(s). This process is activated when the software product undergoes modifications to code and associated documentation due to a problem or the requirement for improvement or adaptation. The objective is to modify an existing software product while preserving its integrity.

This process briefly consists of the following activities (see also: [Europ.Coop. for Space Standardisation (1996)])

- **Problem and modification reporting procedure:** We shall establish procedures for receiving, recording and tracking problem reports and modification requests from the PRISM users. Whenever problems are encountered, they shall be recorded, documented and made 'public' in order to be aware of the 'potential' problem related to a specific PRISM version. For the tracking mechanism we propose the use of the free software RT (Request Tracker). *"RT is an industrial-grade ticketing system. It lets a group of people intelligently and efficiently manage requests submitted by a community of users. RT is used by systems administrators, customer support staffs, NOCs, developers and even marketing departments at over a thousand sites around the world."* For more information please see: (<http://www.fsck.com/projects/rt/>) or (<http://freshmeat.net/projects/requesttracker/>).
- **Problem and modification analysis:** The maintainer shall analyse the problem report or modification requests for its impact on the organization, the existing system, and the interfacing systems for the following:
  - **type** (e.g. corrective, improvement, preventive, or adaptive to new environment);
  - **scope** (e.g. size of modification, cost involved, time to modify);
  - **criticality** (e.g. impact on performance; scientific results).

The maintainer shall be able to reproduce or verify the problem. Based upon the analysis, the maintainer shall develop options for implementing the modification. The maintainer shall document the problem/modification request, the analysis results and implementation options.

- **Implementation of the modifications:** The maintainer shall conduct analysis and determine which documentation, software units, and versions there of shall be modified. The maintainer shall implement the modifications. The requirements of the development proc-

ess shall be supplemented as follows: Test and evaluation criteria for testing and validating the modified as well as the unmodified parts (basic software units, components, ...) of the system shall be defined and documented.

The complete and correct implementation of the new and modified requirements shall be ensured following the usual PRISM Components and System Validation Conventions (as described in section V.4, and V.5).

- **Maintenance review/acceptance:** The maintainer shall conduct a joint Peer Review(s) to determine the integrity of the modified component(s) and system. Upon successful completion of the reviews, a baseline for the change shall be established, defining the importance and impact of the new revision on the entire PRISM system (minor or major revision, main line or branches, etc...).

## Code Reviews

Formal reviews of the code where the code is gone through line-by-line in groups or in pairs has shown to be one of the most effective way to catch errors McConnell (1993). As such it is recommended that component model development teams create a strategy for regularly reviewing the code.

### Possible implementation of Code Reviews

Code reviews can be implemented in many different fashions, but in general they involve having at least one person besides the author go through the written code and examine the implementation both for design and errors. Jones (1986) states that "the average defect-detection rate is only 25 % for unit testing, 35 % for function testing, and 45 % for integration testing. In contrast, the average effectiveness of design and code inspections are 55 % and 60 %. McConnel (1993)] also notes that as well as being more effective in catching errors, code reviews also catch different types of errors than testing does. In addition when developers realize their code will be reviewed they tend to be more careful themselves when programming.

Code reviews can be implemented in different ways:

- **Code validator** Before code is checked into CVS it goes through a "validator" who not only is responsible for testing, and validation of the changes, but also reviews it for design and conformance to code standards.
- **Peer reviews** Before code is checked into CVS a peer developer reviews the changes.
- **Pair programming** All code is developed with two people looking at the same screen (one of the practices of Extreme Programming (Beck (2000))).
- **Formal group walk-through** Code is presented and gone through by an entire group.
- **Formal individual walk-through** Different individuals are assigned and take responsibility to review different subroutines.

## Recommendation

It is recommended that development teams provide both a mechanism to review incremental changes, and also have formal walk-through of important pieces of code as a group. This serves two purposes: the design is communicated to a larger group, and the entire group also reviews the design and implementation. The frequency of the review has to be defined in advance and should in particular consider the criticality of the code in respect to the entire PRISM system.

## References

Mark Baker (ed.),  
Cluster Computing White Paper,  
University of Portsmouth, UK, December 2000  
<http://www.dcs.port.ac.uk/mab/tfcc/WhitePaper>

Kent Beck,  
Extreme Programming Explained; Embrace Change,  
Boston, Mass.: Addison-Wesley, 2000

Barry Cipra,  
Self-tuning' software adapts to its environment,  
Science, 286 p.35, October 1999

David E. Culler et al.,  
Parallel Computer Architecture: A Hardware/Software Approach,  
Morgan Kaufmann Publishers Inc., August 1998

European Cooperation for Space Standardization,  
Space Engineering: Software,  
Technical Report ECCS-Q-40A, ECSS, Requirements and Standard Division, April 1999

European Cooperation for Space Standardization,  
Space Product Assurance: Software Product Assurance,  
Technical Report ECCS-Q-80A, ECSS, Requirements and Standard Division, April 1996

Michael J. Flynn,  
Some computer organizations and their effectiveness,  
IEEE Transactions on Computing, C-21:948-960, 1972

Rupert W. Ford and Graham D. Riley,  
The Met Office FLUME Project - Model Coupling Requirements,  
Manchester Informatics Ltd., The University of Manchester, January 2002

Ian Foster,  
The Grid: A New Infrastructure for 21st Century Science,  
Physics Today, Feb 2002, pp.42 ff

IEEE Standard Glossary of Software Engineering Terminology,  
Technical Report IEEE Std 610.12-1990, Institute of Electrical and Electronic Engineers, December 1990

Capers Jones,  
Programming Productivity,  
New York, New York: McGraw-Hill, 1986

Bill Joy, Ken Kennedy et al.,  
Report to the President: Information technology research: Investing in our future,  
Technical Report, PITAC, President's Information Technology Advisory Committee, Feb. 1999  
<http://www.hpcc.gov/ac/report/>

Angelo Mangili et al.,  
PRISM REDOC III.2: Review of Current and Future HPC Architectures,  
Technical Report , PRISM, June 2002

Steve McConnell,

Code Complete,  
Redmond, Wash.: Microsoft Press, 1993

Steve McConnell,  
Rapid Development,  
Redmond, Wash.: Microsoft Press, 1996

Polcher, J., K. Laval, L. Dumenil, J. Lean and P. R. Rowntree (1996)  
Comparing three land surface schemes used in general circulation models,  
Journal of Hydrology, 180,373-394,.

Polcher *et al.* (1998)  
A proposal for a general interface between land-surface schemes and general circulation models.  
**Global and Planetary Change**, 19: 263-278

Frank S. Preston,  
A peta ops era computing analysis,  
Technical Report CR-1998-207652, March 1998  
<http://techreports.larc.nasa.gov/ltrs/>

James M. Rosinski, David L. Williamson,  
The Accumulation of Rounding Errors and Port Validation for Global Atmospheric Models,  
SIAM Journal on Scientific Computation, Vol. 18, No. 2, March 1997, pp.552-564

Aad J. van der Steen et al.,  
Overview of Recent Supercomputers, 2001  
<http://www.top500.org/ORSC/2001/>

Various sources,  
Science (Mar. 2), Nature (Apr. 11), New York Times (Apr. 20), BBC (Apr. 26) HPC Wire (May 3), 2002



## Glossary

Latest Change August 9 2002

AGCM [Atmosphere General Circulation Model]

**algorithmic coupling interface** : order and exchange frequency of coupling fields between component models (flow chart of exchange)

AOGCM [Atmosphere - Ocean coupled General Circulation Model]

**API [Application Programming Interface]** : interface (calling conventions) by which an application program accesses operating system and other services. An API is defined at source code level and provides a level of abstraction between the application and the kernel (or other privileged utilities) to ensure the portability of the code. An API can also provide an interface between a high level language and lower level utilities and services that were written without consideration for the calling conventions supported by compiled languages. In this case, the API's main task may be the translation of parameter lists from one format to another and the interpretation of call-by-value and call-by-reference arguments in one or both directions.

ARCDI [ARchitecture Choices, detailed Design and Implementation] : part 2 of the PRISM system specification document

(FhG) ASCI [Fraunhofer Gesellschaft, Institute for Algorithms and Scientific Computing]

**ASCII** : basis of character sets used in almost all present-day computers. US-ASCII uses only the lower seven bits (character points 0 to 127) to convey some control codes, space, numbers, most basic punctuation, and unaccented letters a-z and A-Z. More modern coded character sets (e.g., Latin-1, Unicode) define extensions to ASCII for values above 127 for conveying special Latin characters (like accented characters, or German ess-tsett), characters from non-Latin writing systems (e.g. Cyrillic, or Han characters), and such desirable glyphs as distinct open- and close-quotation marks.

CCM [Coupled Climate Model]

CDMS [Climate Data Management System]

**CE dynamic/interactive** : dynamic/ interactive with respect to coupling exchange characteristics

**central repository** : place where information on where things reside can be found; the content is unique

**CF interactive/dynamic** : interactive/dynamic with respect to coupling field characteristics

**CMOS [Complementary Metal Oxide Semiconductor]** : semiconductor fabrication technology using a combination of n- and p-doped semiconductor material to achieve low power dissipation. Any path through a gate through which current can flow includes both n and p type transistors. Only one type is turned on in any stable state so there is no static power dissipation and current only flows when a gate switches in order to charge the parasitic capacitance.

**component model**: numerical model describing a subsystem of the earths climate which can be (or has been) run as a standalone (forced) simulation model

**(computer) configuration:** arrangement of a (computer) system or component defined by the number, nature, and interconnections of its constituents

concurrent by construction:

([http://www.cerfacs.fr/PRISM/COUPLING/sequential\\_concurrent.html](http://www.cerfacs.fr/PRISM/COUPLING/sequential_concurrent.html))

Two models are concurrent by construction if they are sequential by nature but forced to run concurrently. This requires, at a given coupling time-step, that only coupling-data produced at the preceding time-step are used as input

concurrent by nature :

([http://www.cerfacs.fr/PRISM/COUPLING/sequential\\_concurrent.html](http://www.cerfacs.fr/PRISM/COUPLING/sequential_concurrent.html))

Two models are concurrent by nature if, during the same coupling time-step, data produced by one model depend only on coupling data, which are produced previously by the other model and vice versa.

**(experiment) configuration instance :** abstract compact description of an experiment

**(PRISM system) constraints :** what confines/restricts/limits the design choices of the PRISM system (e.g. computer resources, speed of data transfer, existing model designs,...)

**correctness :** (software engineering)

- the degree to which a system or component is free from faults in its specification, design, and implementation
- the degree to which software, documentation, or other items meet specified requirements
- the degree to which software, documentation, or other items meet user needs and expectations, whether specified or not

**coupling configuration :** set of component models and their -> algorithmic coupling interface

**coupling constellation :** -> coupling configuration

**coupler :** software that couples different component models to form a coupled model. The different constituents of the PRISM coupler are: the -> Driver, the -> Transformer, and the -> PRISM System Model Interface Library (PSMILe).

**coupling implementation:** technical way a certain -> coupling configuration of a model is realized (number of executables, parallelism, etc.)

**coupling field/data :** data generated by one component model but used by another one

**data repartitioning :** action of transferring and rearranging coupling data between two models of which data distributions in parallel processes do not match

**debug :** (software engineering) to detect, locate, and correct faults in a computer program

**DEL [Data Exchange Library] :** part of the coupling software PSMILe. The DEL performs data exchanges with appropriate data repartitioning if needed.

(model/experiment) deployment : -> (software) deployment

**(software) deployment :** physical distribution of the system being built, in terms of how functionality (i.e. software components) are distributed among a set of run-time processing nodes/computers.

**design options** : several possibilities to realize the PRISM system responding to the requirements

**driver** : part of the coupling software that controls a coupled model, e.g. launches the models, monitors their execution, etc.

**dynamic analysis** : (software engineering) process of evaluating a system or component based on its behaviour during execution

**dynamic parameters** : parameters which may change during a run (-> global parameter / universal parameter)

**dynamic simulation** : -> PM dynamic, -> CE dynamic, -> CF dynamic

**end-point data exchange** : exchange between a source model and a target model in which the source model produces the data but does not know the target to which it is delivered, and in which the target model requests the data but does not know the source that produces it; the matching is performed by an external entity (e.g. a coupler).

**error** : (software engineering) difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition

ESM [Earth System Model]

**experiment** : collection of tasks running on a supercomputer, defined by the -> component models (model name, release, resolution, options different from default version), the -> algorithmic coupling interface between the component models, the initial state of the models, initial date and length of the simulation, the -> coupling implementation, the target machine, model raw output (if different from default); an experiment can be made up of a series of consecutive (restarted) -> runs

**failure** : (software engineering) inability of a system or component to perform its required functions within specified performance requirements. It is manifested as a fault.

**fault** : (software engineering) incorrect step, process, or data definition in a computer program

**forced simulation/model** : model simulation where input data are specified without interactive feedback to other models. A simulation can be forced with respect to some data and interactive with respect to others

**(G)UI ((Graphical) User Interface )** : use of pictures rather than words to represent the input and output of a program. A program with a GUI runs under some windowing system (e.g. The X Window System, Microsoft Windows). The program displays certain icons, buttons, dialogue boxes, etc. in its windows on the screen and the user controls it mainly by moving a pointer on the screen (typically controlled by a mouse) and selecting certain objects by pressing buttons on the mouse while the pointer is pointing at them. This contrasts with a command line interface where communication is by exchange of strings of text.

**high-level design** : design of a system without regard to implementation issues

HPC [High Performance Computing]

HTTP [Hyper Text Transmission Protocol]

**I/O type 1** -> integrated data processing

**I/O type 2** -> run shell data processing

**I/O type 3** -> post processing

**interactive simulation** : -> PM interactive, -> CE interactive, -> CF interactive

**integrated data processing** : real time data processing performed in the coupled model executables

**Java** : programming language invented by Sun which runs on any platform and supports web services

**JINI** : an open Java architecture network technology that enables developers to create network-centric services that are highly adaptive to change

**Kerberos** : Network security system developed by MIT

LAN [Local Area Network]

**local repository** : copy of a central/main repository with overwrites of local system parameters and addition of user specific parameters

**local transformation** : operation that can be completed in a model without any information from another model, such as finding the maximum value of a field

low/high - end visualization :

[http://prism.enes.org/WPs/WP4a/Meetings/graphic\\_lowhigh.gif](http://prism.enes.org/WPs/WP4a/Meetings/graphic_lowhigh.gif)

**'low-end' visualization** means to select a data set on the remote server, to filter the data, assign the algorithms, operations, plot style and plot format etc. within a web interface to generate the desired output plot file. The plot file could be displayed online using a Java applet or will be transferred to the local client to be displayed offline using a graphic browser. 'low-end' graphics would be used within PRISM to allow for easy generation of standard plots' for (quasi online) quality control, comparison of different runs or models and also to a limited extend for presentation and publication.

**'high-end' graphics** means also to select a data set on the remote server, but to transfer the data file directly to the local client. The user should be able to use the whole data set to assign the algorithms, operations and plot style offline using a -> (G)UI. The GUI should be able to create animations, 3D graphic, video sequences and high performance graphic for presentations (slides etc.). 'high-end' would be used to explore the data rather than just presenting it.

LS(S) [Land Surface (Scheme)]

MAC [Message Authentication]

**main repository** : -> central repository

**middleware** : connectivity software consisting of a set of enabling services that allow multiple processes running on one or more machines to interact across a network. Middleware is essential for migrating mainframe applications to client/server applications and for providing for communication across heterogeneous platforms. This technology has evolved during the 1990s to provide for interoperability in support of the move to client/server architectures. The most widely publicized

middleware initiatives are the Open Software Foundation's Distributed Computing Environment (DCE) , the Object Management Group's Common Object Request Broker Architecture (CORBA), and Microsoft's COM/DCOM.

**model output diagnostics** : processing algorithms for raw data

**model raw (output) data** : data produced by a component model at runtime

**MPMD [Multiple Program Multiple Data]** : program running on multiple processors which perform different instructions

MPP [Massive Parallel Processors]

**NetCDF [Network Common Data Form]** : I/O library for binary machine independent data files, suitable for developing a self describing logical data format

**non-local transformation** : operation that requires information from another model, such as interpolation

**NOS [Networking Operating System]**: operating system, which includes software to communicate with other computers via a network. This allows resources such as files, application programs, and printers to be shared between computers. UNIX systems have these capabilities.

OC [Ocean bio-geo-Chemistry model]

OGCM [Ocean General Circulation Model]

**ORB [Object Request Broker]** : -> middleware technology that manages communication and data exchange between objects. ORBs promote interoperability of distributed object systems because they enable users to build systems by piecing together objects- from different vendors- that communicate with each other via the ORB. The implementation details of the ORB are generally not important to developers building distributed systems. The developers are only concerned with the object interface details. This form of information hiding enhances system maintainability since the object communication details are hidden from the developers and isolated in the ORB.

**parallel partitioning** : describes which parts of the global data are respectively locally treated by the different processes of a parallel model

**physical coupling interface** : physical description of the nature of coupling information exchanged between component models

**PM dynamic / interactive** : interactive with respect to process management

PMIOD [Potential Model Input & Output Description] (-> SMIOC)

**post processing** : off-line processing, typically run on processing servers or workstations using data typically restored from an archive.

**PRISM component model** : -> component model of one of the climate components defined by PRISM, which is accepted by the PRISM community

**PRISM (model) administrator** : the person who makes the quality control of an update to a PRISM (model) code and introduces the updates to the default version

**PRISM (model) developer** : someone who makes changes to a PRISM (model) code that will be introduced to the default (PRISM model) version

**PRISM user** : someone who uses the PRISM System to assemble, deploy, run, and supervise a coupled model

**PSMILe [PRISM System Model Interface Library]** : library linked to the component models interfacing them with the rest of the coupled model. The PSMILe includes the -> Data Exchange Library, the I/O library, and some coherence check and local transformation routines.

**pseudo component model** : model which can replace a 'real' component model in a simulation. It reads forcing data from disk instead of calculating them. It might use bulk formulas for the transformation of data. When pseudo models are involved in a PRISM coupled model then the model is (at least partially) -> forced.

**point-wise transformation** : operation that can be completed on each grid point without any external information, neither from the model neighbouring grid points, neither from another model; examples are time averaging or summation of coupling fields given on the same grid

**REDOC [REquirements, Design Options and Constraints]** : part 1 of the PRISM system specification document

**repository** : shared database of project information (as a passive kernel for information integration), -> central/main repository, -> local repository

**RISC** : processor whose design is based on the rapid execution of a sequence of simple instructions rather than on the provision of a large variety of complex instructions (as in a Complex Instruction Set Computer).

RL [Request Listener]

RT [Request Tracker]

**RMI [Remote Method Invocation]** : -> SW technique for invoking code on a remote host

**run** : ensemble of tasks defined by a configuration process. The tasks are under the control of a scheduler coordinating the execution of the -> tasks while preserving any dependencies between them. An -> experiment can comprise several consecutive (restarted) runs

**run shell data processing** : data processing done as an afterburner within the coupled system (e.g. under -> SMS control) and typically performed before data is sent to the archive. This may be run on the supercomputer or suitable server

SCC [Specific Coupling Configuration]

sequential by nature :

([http://www.cerfacs.fr/PRISM/COUPLING/sequential\\_concurrent.html](http://www.cerfacs.fr/PRISM/COUPLING/sequential_concurrent.html))

Two models are sequential by nature if the first model necessarily waits while the second model is running, and vice versa. The sequence is imposed by the exchange of coupling fields

sequential by construction :

([http://www.cerfacs.fr/PRISM/COUPLING/sequential\\_concurrent.html](http://www.cerfacs.fr/PRISM/COUPLING/sequential_concurrent.html))

Two models are sequential by construction if they are concurrent by nature but forced to run se-

quentially. This requires, at a given time-step, that coupling data produced at the preceding time-step are used as input.

SI [Sea Ice model]

**S/Key [SecureKey]** : one-time password system

SMIOC [Specific Model Input and Output Configuration]

SMP [Symmetric Multi-Processing]

**SOAP** : lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol

**sockets** : the Berkeley Unix mechanism for creating a virtual connection between processes. Sockets interface Unix's standard I/O with its network communication facilities. They can be of two types, stream (bi-directional) or datagram (fixed length destination-addressed messages). The socket library function `socket()` creates a communications end-point or socket and returns a file descriptor with which to access that socket. The socket has associated with it a socket address, consisting of a port number and the local host's network address.

**SPMD [Single Program Multiple Data]** : program running on multiple processors performing all the same instructions

**SSH [Secure Shell]** : secure remote access protocol pioneered by BSD enabling remote logins

**SSL [Secure Socket Layer]** : secure communication protocol invented by Netscape

SSW [PRISM System Specification Workgroup]

**static parameter** : parameters which may not change during a run (-> dynamic parameter)

SW [Soft Ware]

**TCP/IP [Transmission Control Protocol over Internet Protocol]** : de facto standard Ethernet protocol incorporated into 4.2 BSD Unix. TCP/IP was developed by DARPA for internetworking and encompasses both network layer and transport layer protocols. While TCP and IP specify two protocols at specific protocol layers, TCP/IP is often used to refer to the entire DoD protocol suite based upon these, including telnet, FTP, UDP and RDP.

**Testing**: (software engineering) process of analysing a software item to detect the differences between existing and required conditions (-> bugs) and to evaluate the features of the software items

**Static analysis**: (software engineering) process of evaluating a system or component based on its form, structure, content, or documentation

**Task**: individual job step of a -> run that needs to be executed; e.g. execution of a component model, of the coupler (if it is a separate process), model raw output diagnostics, data archiving, etc.

Time horizon for 'future model developments': order 10 years

**Toy component model**: simple model containing all PRISM functionalities, but no 'real' physics. The toy model can be used as implementation guide for the adaptation of existing models (including -> pseudo models) to the PRISM software and for testing purposes.

**Toy PRISM system:** coupled model consisting of a toy model for each PRISM component model coupled to the PRISM coupler. The toy model shall be able to test every function of the PRISM software, as well as algorithmic and physical interfaces.

**Transformer:** part of the PRISM coupling software that performs all transformations required on the coupling fields between two component models

UDDI [Universal Description, Discovery and Integration]: enables dynamic lookup and advertising of services

**Universal parameters:** parameters, which do not change during an experiment and must be consistently defined for all PRISM components

**Validation:** (software engineering) process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements

**Verification:** (software engineering)

- Process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase
- formal proof of program correctness

WAN [Wide Area Network]

### **WS [Web Services]**

**WSDL:** XML format for describing network services as a set of endpoints operating on messages containing either document oriented or procedure oriented information

**XML [eXtensible Markup Language]:** universal format for structured documents and data on the Web. It is a human-readable, machine-understandable, general syntax for describing hierarchical data, applicable to a wide range of applications. Custom tags enable the definition, transmission, validation, and interpretation of data between applications and between organizations

**X509:** standard for certificates used by SSL authentication and encryption



*Appendices:*

<b>PRISM</b> .....	1
<b>SYSTEM SPECIFICATION</b> .....	1
<b>HANDBOOK</b> .....	1
<b>VERSION 1.0</b> .....	1
<b>CONTRIBUTORS</b> .....	7
<b>FOREWORD TO THE PRISM SYSTEM SPECIFICATION</b> .....	9
<b>CONTENT</b> .....	11
<b>EXECUTIVE SUMMARY</b> .....	15
<b>REFERENCES</b> .....	191
<b>GLOSSARY</b> .....	193
<b>APPENDICES:</b> .....	201
<b>APPENDICES TO REDOC I.2.:</b> .....	202
ATMOSPHERE	202
ATMOSPHERIC CHEMISTRY	206
LAND SURFACES	210
OCEAN	213
SEA-ICE	217
OCEAN BIOGEOCHEMISTRY	219
REGIONAL MODEL INPUTS	222
<b>APPENDIX TO REDOC I.3:</b> .....	226
REQUIREMENTS SUMMARY TABLE	226
ACKNOWLEDGEMENTS	232
COUPLING FIELDS	232
<b>TABLES</b> .....	235
<b>FIGURES</b> .....	237

*Appendices to REDOC I.2.:*

## *Atmosphere*

Version 1.0 (Mar 11th 2002)

*SERGE PLANTON*

**Purpose:** List all inputs required for an atmosphere model component to solve its prognostic/diagnostic equations. The list of inputs should be model independent and solely based on the modeled aspects of the physics of the earth system as we know it today, and on those aspects we anticipate to become important in future modeling studies. See main REDOC I.2 text for explanations.

Field	Process	Unit	Time scale	Issues	Rating	Origin
Sea Surface Temperature	Emission of long-wave radiation and turbulent fluxes (latent, sensible, momentum,...)	K	1h		E	O
Surface albedo	Reflection of diffuse solar radiation	No	1h	(I): albedo for direct solar radiation may be calculated in the atmosphere component	E	O
Surface albedo	Reflection of diffuse solar radiation	No	1h	(i) + (ii): for different frequency bands	M	O
Surface emissivity	Emission of long-wave radiation	No	1h		M	O
Surface roughness length	Turbulent fluxes	m	1h		E	O
Surface temperature	Emission of long-wave radiation and turbulent fluxes	K	Time step of Atmosphere component	(iii): different ice types may be distinguished	E	SI
Surface albedo	Reflection of diffuse solar radiation	No	Time step of Atmosphere component	(i) + (iii)	E	SI
Surface albedo	reflection of diffuse solar radiation	No	Time step of Atmosphere component	(i) + (ii) + (iii)	M	SI
Surface emissivity	emission of long-wave radiation	No	Time step of Atmosphere component	(iii)	M	SI
Surface roughness length	turbulent fluxes	m	Time step of Atmosphere component	(iii)	E	SI
Ice concentration	turbulent fluxes	No	Time step of Atmosphere component	(iii)	E	SI
Volumic mixing ratio or mass of different species (CO <sub>2</sub> , CH <sub>4</sub> , N <sub>2</sub> O, O <sub>3</sub> , ClO, ...)	radiation and chemical-transport processes	mol/mol or Kg	6h (loose coupling) to time step of Atmosphere component (tight coupling)	(iv): to account for the treatment of some chemical-transport processes by the Atmosphere component	E	AC
Size spectrum characteristics of	radiation and chemical-transport processes	Kg/(size classes	6h (loose coupling)	(iv)	E	AC

Field	Process	Unit	Time scale	Issues	Rating	Origin
aerosols (sul-fates, black carbon, ...)		or distribution modes)	to time step of Atmosphere component (tight coupling)			
Second-order moments of volumic mixing ratio or mass of different species or size spectrum characteristics of aerosols	chemical-transport processes	mol/mol or Kg or Kg/(size classes or distribution	6h (loose coupling) to time step of Atmosphere component (tight coupling)	to account for the treatment of transport or sub-grid scale chemical processes by the Atmosphere component	M	AC
Land Surface temperature	emission of long-wave radiation and turbulent fluxes	modes)	time step of Atmosphere component	(iv): different surface conditions may be considered within each grid mesh	E	LS
Surface albedo	reflection of diffuse solar radiation	No	time step of Atmosphere component	(i) + (iv)	E	LS
Surface albedo	reflection of diffuse solar radiation	No	time step of Atmosphere component	(i) + (ii)+ (iv)	M	LS
Surface emissivity	emission of long-wave radiation	No	time step of Atmosphere component	(iv)	M	LS
Latent and sensible surface heat fluxes	turbulence in the atmospheric boundary layer	W/m <sup>2</sup>	time step of Atmosphere component	(iv)	E	LS
Surface roughness length	turbulent fluxes	m	time step of Atmosphere component	(iv)	E	LS
Displacement height	turbulent fluxes	m	time step of Atmosphere component	(iv)	D	LS
Transfer coefficients for heat fluxes	turbulent fluxes	No	time step of Atmosphere component	(iv)	E	LS

Field	Process	Unit	Time scale	Issues	Rating	Origin
			ment			
Actual over potential evaporation	latent heat flux (and evapotranspiration)	No	time step of Atmosphere compone	(iv)	E	LS
Surface humidity	bare soil latent heat flux (and evaporation)	No	time step of Atmosphere compone	(iv)	E	LS

CONTRIBUTORS:

nt

**Note:** This list, classified according to the different atmosphere models, corresponds to the participants to Work Package 3b of PRISM. Please check it and correct/complete this list if needed.

1. ARPEGE-CLIMAT (ECMWF + METEO-FRANCE/CNRM)

S. BELAMARI (METEO-FRANCE/CNRM)

A. BRAUN (METEO-FRANCE/CNRM)

M. DEQUE (METEO-FRANCE/CNRM)

M. MILLER (ECMWF)

J. PH. PIEDELIEVRE (METEO-FRANCE/CNRM)

D. SALAS Y MELIA (METEO-FRANCE/CNRM)

2. ECHAM (MPI)

L. KORNBLUEH (MPI/IMET)

S. LEGUTKE (MPI/MAD)

A. NAVARRA (ING)

E. ROECKNER (MPI/IMET)

3. LMDZ (IPSL/LMD)

P. BRACONNOT (IPSL/LSCE)

L. FAIRHEAD (IPSL/LMD)

M.A. FOIJOLS (IPSL)

J.-Y. GRANDPEIX (IPSL/LMD)

F. HOURDIN (IPSL/LMD)

H. LE TREUT (IPSL/LMD)

L. LI (IPSL/LMD)

J. POLCHER (IPSL/LMD)

4. UNIFIED MODEL (UKMO)

M. CARTER (UKMO)

J. GREGORY (UKMO)

D. GRIGGS (UKMO)

T. JOHNS (UKMO)

J. MITCHELL (UKMO)

J. SLINGO (UNIV. READING)

# *Atmospheric Chemistry*

Version 1.0 (March 17th 2002)

Martin Schultz (MPI-M)

**Purpose:** List all inputs required for an atmosphere chemistry model component to solve its prognostic/diagnostic equations. The list of inputs should be model independent and solely based on the modeled aspects of the physics of the earth system as we know it today, and on those aspects we anticipate to become important in future modeling studies. See main REDOC 1.2 text for explanations.

Notes:

As discussed during the workshop, it is inherently difficult to draw a clear line between atmospheric chemistry and the physical atmosphere model. Hence, this table includes some quantities, which are traditionally computed within an atmospheric chemistry model although they are actually physical parameters. These include AC in the originator field.

Added dimensionality of fields

Origin in parentheses means: future option, currently probably not realised in any model

Field	Process	Unit	Dims	Time scale	Issues	Rating	Origin
Dry air temperature		K	3	min	probably uncritical w.r.t. t+1 or t-1	E	A
surface temperature	deposition/emissions	K	2	min		E	A (LS)
potential temperature		K	3	min	useful diagnostic	D	A
virtual pot. temperature		K	3	min	useful diagnostic	D	A
full level pressure		Pa	3	min	probably uncritical w.r.t. t+1 or t-1	E	A
half level pressure		Pa	3	min	probably uncritical w.r.t. t+1 or t-1	E	A
geopotential height			3	min	probably uncritical w.r.t. t+1 or t-1	E	A
air density	conversion mass<-> mixing ratio	molecules/cm <sup>3</sup> /s	3	min	probably uncritical w.r.t. t+1 or t-1	E	A, AC
specific humidity		kg(H <sub>2</sub> O)/kg(dry air)	3	min	maybe critical (t+1) in sophisticated scavenging schemes	E	A
relative humidity	used for photolysis, emissions, aerosol dynamics	fraction	3	min	alternatively: saturation humidity	E	A
land fraction	deposition, emissions	fraction	2	month	maybe shorter timescale (days) if snowfall is taken into account explicitly	E	A (LS)
land mask	deposition, emissions	flag	2	year	simpler alternative for land fraction	E (D)	DS, A
(sea)ice fraction	deposition, emissions	fraction relative to ocean fraction	2	day	currently rather monthly timescale	E	DS, A
vegetation fraction	deposition	fraction relative to land fraction	2	month		E	A, LS (TB, DS)
leaf area index	deposition	index	2	day	currently rather monthly timescale	E	A (TB, DS)
convective precipitation	emissions	mm/day	2	min		E	A
Large-scale precip.	emissions	mm/day	2	min		E	A
evaporation	scavenging	??	3	min	critical w.r.t. to time step	E	A
condensation	scavenging, aerosol dynamics	??	3	min	critical w.r.t. to time step	E	A
surface albedo	photolysis	fraction	2	hour	potential improvement using spectrally resolved albedo	E	A, DS
UV albedo	photolysis	fraction	2	hour		D	A, DS
zenith angle	photolysis	angle	2,3	min		E	A
actinic UV and visible flux	photolysis	photon-flux	3	min	spectrally resolved (7+ bands)	E	A, AC
surface net radiation	deposition, emissions	W/m <sup>2</sup>	2	min	maybe augmented by PAR (photosynthetically active radiation)	E	A

Field	Process	Unit	Dims	Time scale	Issues	Rating	Origin
cloud fraction	photolysis	fraction	3	min	potential improvements: distinction between maximum overlap and random overlap; use of beta distribution	E	A
cloud water	photolysis, scavenging,	kg(H <sub>2</sub> O)	3	min	time critical	E	A
cloud ice	photolysis, scavenging,	kg(H <sub>2</sub> O)	3	min	time critical	E	A
potential vorticity	(lightning)	??	3	min	useful diagnostic (stratosphere-troposphere exchange)	D	A
convective mass flux	lightning (and transport diagnostic if multiplied by tracer mixing ratio)	??	3	min	not yet implemented in many models	E	A
maximum updraft velocity	lightning	m/s	2	min		E	A
max. cloud top height	lightning	m	2	min	alternative to wmax	E	A
depth of cloud charging zone	lightning	m	2	min	charging zone defined as mixed phase zone. Typically diagnosed from temperature threshold	E	AC, A
flash frequency	lightning	flashes/min	2	min		E	AC, A
wet deposition	scavenging	??	3	min		D	AC, A
flex	deposition	??	??	min		E	A
axial drag coefficient	deposition	??	??	min		E	A
Richardson number	deposition	??	2	min		E	A
surface roughness	deposition	??	2	min	maybe improvement for anisotropic conditions?	E	A
snow cover	deposition	m	2	hour		E	A
soil moisture stress	deposition, emissions	??	2	hour		E	A, LS
10m wind components	deposition, air-sea-exchange	m/s	2x2	min		E	A
temperature of top soil layer	emissions	K	2	hour		E	A, LS
field capacity	deposition	??	2	hour		E	A, LS
overhead ozone column	photolysis	DU	2	hour	depends on altitude range of model ensemble	E	DS, A
vegetation type and fuel	emissions (biomass burning)	index, t/ha	N*2	day	currently used in preprocessing step	D	TB, LS



Field	Process	Unit	Dims	Time scale	Issues	Rating	Origin
load							

CONTRIBUTES FROM:

LAURENS GANZVELD (MPI-C),

ROLF SANDER (MPI-C),

MARTIN SCHULTZ (MPI-M),

PHILIP STIER (MPI-M),

PETER VAN VELTHOVEN (KNMI)

# Land Surfaces

Version 1.0 (March 11th 2002)

JAN POLCHER

**Purpose:** List all inputs required by land-surface model component to solve its prognostic/diagnostic equations. The list of inputs should be model independent and solely based on the modeled aspects of the physics of the earth system as we know it today, and on those aspects we anticipate to become important in future modeling studies. See main REDOC I.2 text for explanations.

The present list is derived from the document written during the [PILPS-4c project](http://www.lmd.jussieu.fr/~polcher/PILPS4c/main.html) (See the PILPS-4c paper under <http://www.lmd.jussieu.fr/~polcher/PILPS4c/main.html>)

\* It goes beyond what is described there as since then a number of developments have taken place. Some choices are proposed but they need to be re-discussed:

The atmospheric model should compute the diffusion coefficients for the surface layer. This assumes that turbulence in the surface layer is entirely treated by the atmospheric components.

The land-surface scheme also deals with the momentum flux. The reason is that it allows taking advantage of the sub-grid scale variability described in the land-surface scheme. This also means that the atmosphere will have to take care of the momentum flux induced by the orography.

## Radiation input

Field	Process	Units	Time scale	Issues	Rating	Origin
Net short-wave radiation	Energy input to the surface energy balance	W/m <sup>2</sup>	> hour	The variable should be averaged over all spectral bands.	E	A
Downward long-wave radiation	Energy input to the surface energy balance	W/m <sup>2</sup>	> hour	The variable should be averaged over all spectral bands.	E	A
Downward solar flux	Calculation of albedo	W/m <sup>2</sup>	> hour	These variables should have one dimension, which discretizes the simulated spectral bands. It should contain at least the 2 bands	D	A
Fraction of diffuse radiation	Calculation of albedo	-	> hour	Ased today for each spectral band needs to be provided.	D	A
Solar Zenith angle	Calculation of albedo	deg	> hour	-	D	A

### Inputs for the hydrological cycle

Field	Process	Units	Time scale	Issues	Rating	Origin
Rain fall	Closing the water cycle	kg/m <sup>2</sup> /s	> hour	It is essential to separate the liquid from the snow precipitation.	E	A
Snow fall	Closing the water cycle	kg/m <sup>2</sup> /s	> hour	-	E	A
Sub-grid scale variance of rain fall	Representing interception and infiltration	kg/m <sup>2</sup> /s	> hour	-	D	A
Sub-grid scale variance of snow fall	Representing interception	kg/m <sup>2</sup> /s	> hour	-	D	A

processes

### Inputs for the calculation of turbulent fluxes

Field	Process	Units	Time scale	Issues	Rating	Origin
Lowest atmospheric air temperature	Simulation of the turbulent sensible heat flux	K	minutes	-	E	A
Lowest atmospheric air specific humidity	Simulation of the turbulent latent heat flux	g/g	minutes	-	E	A
Lowest atmospheric CO <sub>2</sub> concentration	Simulation of the turbulent carbon flux	ppm	minutes	-	E	A
Surface eddy diffusivity coefficient for momentum	Simulation of surface stress not related to orography	m/s	minutes	-	E	A
Surface eddy diffusivity coefficient for the sensible heat flux	Simulation of the turbulent sensible heat flux	m/s	minutes	-	E	A
Surface eddy diffusivity coefficient for the latent heat	Simulation of the turbulent latent heat flux	m/s	minutes	-	E	A
Surface eddy diffusivity coefficient for CO <sub>2</sub> flux	Simulation of the turbulent carbon flux	m/s	minutes	-	E	A
Sensitivity of air temperature to surface fluxes	Simulation of the turbulent sensible heat flux	J/kg	minutes	This is needed for an implicit resolution of the surface energy balance. The theory is explained in the <a href="#">PILPS-4c paper*</a> The coefficients A and B are needed.	E	A
Sensitivity of air specific humidity to surface fluxes	Simulation of the turbulent latent heat flux	g/g	minutes	Explained in <a href="#">PILPS-4c paper*</a> As above these are two values we need.	E	A
Sensitivity of atmospheric CO <sub>2</sub> to surface fluxes	Simulation of the turbulent carbon flux	ppm	minutes	As above	D	A
Surface pressure	Calculation of saturated humidity	hPa	> hour	-	E	A

## *Input for computing chemical fluxes*

It still needs to be discussed if the processes, which determine the surface fluxes of chemical species, need to be treated within the land-surface model or if it is sufficient to provide some of the controlling variables.

Key to this discussion is the strength of the interaction, which exists between physical and biogeochemical processes represented in land-surface models on the one hand and chemical processes on the other.

<b>Field</b>	<b>Process</b>	<b>Units</b>	<b>Time scale</b>	<b>Issues</b>	<b>Rating</b>	<b>Origin</b>
Concentrations of all chemical species used in the AC.	Deposition and emission	-	< 1 hour	None of these fluxes seem to be tightly enough linked to the turbulence in the surface layer to have to solve them with the PBL, as is done for water or temperature.	D	AC

# Ocean

Version 3.1 (May 27th 2002)

Eric Guilyardi

**Purpose:** List all inputs required by ocean model component to solve its prognostic/diagnostic equations. The list of inputs should be model independent and solely based on the modeled aspects of the physics of the earth system as we know it today, and on those aspects we anticipate to become important in future modeling studies. See main REDOC I.2 text for explanations.

Energy fluxes	Process	Unit	Time scale	Issues	Rating	Origin
Short wave flux	Solar penetration	W/m <sup>2</sup>	1-3h	2D - which frequency band through leads under sea ice?	E	A&SI
Long-wave	Heat budget of ocean	W/m <sup>2</sup>	1-3h	2D	E	A
Latent heat release due to solid precipitations	Heat budget of ocean	W/m <sup>2</sup>	1-3h	2D	E	A
Latent heat release due to icebergs melting	Heat budget of ocean	W/m <sup>2</sup>	1-3h	2D	E	? (LS)
Latent heat	Heat budget of ocean	W/m <sup>2</sup>	1-3h	2D	E	A
Sensible heat	Heat budget of ocean	W/m <sup>2</sup>	1-3h	2D	E	A
Sensible heat due to precipitations	Heat budget of ocean	W/m <sup>2</sup>	1-3h	2D	E	A
Sensible heat due to runoff	Heat budget of ocean	W/m <sup>2</sup>	1-3h	2D	D	LS
Heat flux at base of sea ice	Heat budget of ocean	W/m <sup>2</sup>	ocean time step	2D - where sea ice is present	E	SI
Geothermal heat flux	Heat budget of ocean	W/m <sup>2</sup>		2D	U	?

Mass fluxes	Process	Unit	Time scale	Note/Issues	Rating	Origin
Liquid precipitation	Salinity budget at surface / sea level	kg/m <sup>2</sup> /s	3h	2D (associated heat flux)	E	A
Solid precipitations	Salinity budget at surface / sea level	kg/m <sup>2</sup> /s	3h	2D (associated heat flux)	E	A
Evaporation	Salinity budget at surface / sea level	kg/m <sup>2</sup> /s	3h	2D	E	A
River outflow	Salinity budget at surface / sea level	m <sup>3</sup> /s	3h	2D	E	LS
Iceberg melting	Salinity budget at surface / sea level	kg/m <sup>2</sup> /s	24h	2D	E	?
Melting of sea-ice	Salinity budget at surface / sea level	kg/m <sup>2</sup> /s	3h	2D - where sea ice is present	E	SI
Formation of sea-ice	Salinity budget at surface / sea level	kg/m <sup>2</sup> /s	3h	2D - where sea ice is present	E	SI
Ice shelf melting	Salinity budget at surface / sea level	m <sup>3</sup> /s	24h or more	2D	D	?
Snow melt at surface of sea ice	Salinity budget at surface / sea level	kg/m <sup>2</sup> /s	3h	2D - where sea ice is present	E	SI

Momentum fluxes	Process	Unit	Time scale	Note/Issues	Rating	Origin
Wind stress over open ocean	momentum balance	N/m <sup>2</sup>	1h	2D - Vector - link to wave model ?	E	A
Ocean sea ice stress	momentum balance	N/m <sup>2</sup>	ocean time step	2D - Vector (where sea ice is present)	E	SI
Wind "power" or "mixing"	Vertical mixing	(m/s) <sup>3</sup>	1h	2D - Input to vertical mixing U <sup>3</sup> Issue : how to handle time averaging and non-linearity ? See OPYC technique.	E	A
2d Ocean wave spectrum	mixing through wave breaking, bubble formation	m <sup>4</sup>	3h	wave model currently not part of PRISM	M	?

Salinity fluxes	Process	Unit	Time scale	Note/Issues	Rating	Origin
Salt flux from sea ice	Salt budget of ocean	kg/m <sup>2</sup> /s	ocean time step	2D	E	SI
Salt flux from atmosphere	Salt budget of ocean	kg/m <sup>2</sup> /s	3h	2D - from foam (wave model?)	D	A
Salt flux from continents	Salt budget of ocean	kg/m <sup>2</sup> /s		2D	U	LS

Others	Process	Unit	Time scale	Note/Issues	Rating	Origin
Ocean colour	Vertical distribution of solar flux	?	Given by OB	3D field, or vertical profile of solar penetration, or optical properties of seawater or extinction coefficients?	E	OB
Local gravity	Density driven processes	m <sup>2</sup> /s	Constant	Same as other components	E	?
Atmospheric MSL pressure	BC on horizontal pressure gradient	hPa	3h	2D	D	A
Ice thickness	Volume of submerged ice	m	Ocean time step	2D - needed to know volume of water in upper layer	E	SI
Snow on sea ice	Volume of submerged ice	m	Ocean time step	2D - needed to know volume of water in upper layer	E	SI

Notes :

1. Tidal energy and mixing will probably remain an *internal* forcing of the ocean GCM for the next 5-10 years. Data might need to be read from a file but not from another earth system component.
2. Geothermal heat flux will also probably remain and *internal* forcing.

Regional or nested coupling (ocean variables only):

Regional or nested coupling	Dimension of field	Unit	Time scale	Note/Issues	Rating	Origin
Potential temperature	strip of data at lateral boundary or 3D if nested	K or C ?	Time step (1h)		D	OGCM
Salinity	strip of data at lateral boundary or 3D if nested	PSU	Time step		D	OGCM
Zonal velocity	strip of data at lateral boundary or 3D if nested	m/s	Time step		D	OGCM
Meridional velocity	strip of data at lateral boundary or 3D if nested	m/s	Time step		D	OGCM
Sea surface height	strip of data at lateral boundary or 3D if nested	m	Time step		D	OGCM

*CONTRIBUTORS:*

*AS TENTATIVELY IDENTIFIED FROM LES DIABLERET MEETING (JUNE 2001)*

*THIERRY FICHEFET AND ERIC GUILYARDI (7/2/2002)*

*GURVAN MADEC, MIKE BELL, RALF DÖSCHER, RENÉ REDLER, HEIKO JANSEN, JOHANN JUNGCLAUS,*

*LAURENCE FLEURY, PIERRE-PHILIPPE MATHIEU (11/03/2002)*

*GERBRAND KOMEN, ADRIAN NEW (27/05/2002)*



# Sea-Ice

Version 2.0 (Feb 18th 2002)

Helge Drange

**Purpose:** List all inputs required by sea-ice model component to solve its prognostic/diagnostic equations. The list of inputs should be model independent and solely based on the modeled aspects of the physics of the earth system as we know it today, and on those aspects we anticipate to become important in future modeling studies. See main REDOC I.2 text for explanations.

Energy Fluxes and Related Quantities	Processes	Unit	Time scale	Note/Issues (sim=sea ice model)	Rating	Origin
Direct solar flux	Solar penetration	W/m <sup>2</sup>		which frequency band?	E	
Diffuse solar flux	Solar penetration	W/m <sup>2</sup>		as above, needed by biochemistry	E	
Surface albedo of ocean	Heat budget	-		Needed for leads	E	
Incoming long wave radiation	Heat budget	W/m <sup>2</sup>		over snow, sea ice and ocean	E	
Outgoing long wave radiation	Heat budget	W/m <sup>2</sup>		-, computed by sim?	E	
Sensible heat flux	Heat budget	W/m <sup>2</sup>		Over snow, sea ice and ocean	E	
Sensible heat flux due to precipitation	Heat budget	W/m <sup>2</sup>		Over snow, sea ice and ocean	E	
Latent heat flux	Heat budget	W/m <sup>2</sup>		Over snow, sea ice and ocean	E	
Derivative of the net non-solar heat flux over sea ice	Heat budget	W/m <sup>2</sup>			E	
Latent heat release associated with ice berg melting	Heat budget	W/m <sup>2</sup>			D	
Sea surface temperature	Heat budget	K			E	
Sea surface salinity	Heat budget	psu		For initial ice salinity, and freezing point of seawater	E	

Fresh Water Fluxes	Process	Unit	Time scale	Note/Issues	Rating	Origin
Precipitation, including snow	Snow/ice mass	kg/m <sup>2</sup> /s			E	A
Evaporation	Snow/ice mass	kg/m <sup>2</sup> /s			E	A
River outflow/ice berg	Ice mass	m <sup>3</sup> /s			D	LS, ?

Momentum Fluxes	Process	Unit	Time scale	Note/Issues	Rating	Origin
Wind stress over sea ice	Momentum balance	N/m <sup>2</sup>		Vector	E	A
Wind stress over open water	Momentum balance	N/m <sup>2</sup>		Vector	E	A
Ocean stress	Momentum balance	N/m <sup>2</sup>		Vector	E	O

Others	Process	Unit	Time scale	Note/Issues	Rating	Origin
Ice bergs/inland ice	Mass and momentum balance					

• **CONTRIBUTIONS:**

AS DISCUSSED AT THE LES DIABLERET MEETING (JUNE 2001), NOTABLY THIERRY FICHEFET AND DAVID SALAS



# Ocean Biogeochemistry

Version 1.1 (Feb 19th 2002)

Corinne Le Quéré

**Purpose:** List all inputs required by ocean biogeochemistry model component to solve its prognostic/diagnostic equations. The list of inputs should be model independent and solely based on the modeled aspects of the physics of the earth system as we know it today, and on those aspects we anticipate to become important in future modeling studies. See main REDOC I.2 text for explanations.

Gas and Particle Fluxes	Unit	Dimension	Note/Issues	Rating	Provider
Wind speed	m/s	2D		E	Atmosphere
Variance of the wind speed	m/s	2D	For off-line or degraded simulations only	E	Atmosphere
Ice extent	Fraction of grid box	2D		E	Ice
Surface atmospheric concentration of gases (i.e. CO <sub>2</sub> , <sup>14</sup> CO <sub>2</sub> , <sup>13</sup> CO <sub>2</sub> , N <sub>2</sub> O, O <sub>2</sub> , <sup>18</sup> OO, <sup>17</sup> OO, DMS)	Variable (ppm, ppb or permil depending on the tracer)	2D	Not all gases are needed depending on the experiment	E	Atmosphere
Wet deposition of particles	g/m <sup>2</sup> /s	2D	Can be computed from the product of precipitation and atmospheric concentration of	E	Atmosphere, Atmospheric chemistry
Dry deposition of particles	g/m <sup>2</sup> /s	2D	particles	E	Atmospheric chemistry
Chemical content of particles (i.e. Fe, Si)	g/g	2D		E	Atmospheric chemistry
Concentration of tracers in ice flow (i.e. Fe, Si)	g/m <sup>3</sup>	2D		D	Ice
Concentration of tracers in river runoff (i.e. DIC, POC)	g/m <sup>3</sup>	2D		D	Land?

Dilution/concentration of tracers	Unit	Dimension	Note/Issues	Rating	Provider
Change in ocean surface elevation	m	2D	For free surface ocean models	E	Ocean
Concentration of the tracers in all water fluxes contributing to the change in surface elevation	mol/m <sup>3</sup>	2D	For free surface ocean models, includes precipitation, evaporation, runoff, ice change	E	Ocean, Ice, Land, Atmosphere
Net water flux	m/s or m <sup>3</sup> /s	2D	For rigid-lid ocean models, includes precipitation, evaporation, runoff, ice change	E	Ocean, Ice, Land, Atmosphere

Marine biology	Unit	Dimension	Note/Issues	Rating	Provider
Surface short wave radiation	W/m <sup>2</sup>	2D	Spectrum and diffusive/non-diffusive desirable	E	Atmosphere
Mixing depth	m	2D	Based on criteria of vertical mixing	E	Ocean
Mixed-layer depth	m	2D	Based on criteria of density	E	Ocean
Density	g/m <sup>3</sup>	3D	To estimate the buoyancy of marine organisms	D	Ocean
Small scale turbulence	m <sup>2</sup> /s	3D	To estimate sub-mm chemical flow between plankton and sea water	D	Ocean

Chemical reactions	Unit	Dimension	Note/Issues	Rating	Provider
Temperature	Celsius	3D	Also used for marine biology and to compute the slope of isopycnals in some models	E	Ocean
Salinity	psu	3D	Also used for marine biology and to compute the slope of isopycnals in some models	E	Ocean

Transport of tracers	Unit	Dimension	Note/Issues	Rating	Provider
Velocity	m/s	3D		E	Ocean
Variance of the velocity	m/s	3D	For off-line or degraded simulations only	E	Ocean
Diffusivity	m <sup>2</sup> /s	3D	Both vertical and lateral	E	Ocean
Gent-McWilliams velocity	m/s	3D		E	Ocean
Slope of isopycnals	no units (ratio)	3D		E	Ocean
Convection index	fraction	3D	Multiple formulations possible	E	Ocean

#### Ocean biogeochemistry outputs

Output	Unit	Dimension	Note/Issues	Rating	User
Gas flux	mol/m <sup>2</sup> /s	2D	CO <sub>2</sub> , DMS, O <sub>2</sub> , N <sub>2</sub> , <sup>13</sup> CO <sub>2</sub> , <sup>14</sup> CO <sub>2</sub> , <sup>17</sup> OO, <sup>18</sup> OO	E	Atmospheric chemistry
Light attenuation by marine particles	W/m <sup>2</sup> or fraction of the surface radiation	3D	Radiation could be split into several wave bands	E	Ocean

CONTRIBUTIONS:

PRISM WORKING GROUP 3G MEETING IN HAMBURG, JANUARY 31ST 2002 (C. LE QUÉRE, E. MAIER-REIMER, O. AUMONT, S. SPALL, S. LEGUTKE). [MINUTES OF THE MEETING AVAILABLE ON HTTP://WWW.BGC.MPG.DE/~CORINNE.LEQUERE/PRISM/MINUTES\\_JAN02.HTML](http://www.bgc.mpg.de/~corinne.lequere/prism/minutes_jan02.html)

## *Regional model inputs*

Version 1.0 (March 27th 2002)

MARKKU RUMMUKAINEN / RALF DOESCHER

**Purpose:** List all inputs required by ocean model component to solve its prognostic/diagnostic equations. The list of inputs should be model independent and solely based on the modeled aspects of the physics of the earth system as we know it today, and on those aspects we anticipate to become important in future modeling studies. See main REDOC I.2 text for explanations.

Concerning coupled models of atmosphere, ocean, sea ice and chemistry, input required for global <-> regional coupling is listed here. Input fields need to be provided for 3D boundary zones along finite lateral boundaries around the regional atmosphere and ocean domains.

The regional component can consist of an atmospheric model only, or an ocean component only. In such cases, global 2D ocean/sea ice information needs to be passed to a regional atmosphere, or global 2D atmospheric information to a regional ocean model. When the regional component consists of a coupled atmosphere-land surface-ocean-sea ice model, some of the regional atmosphere-surface interface is generated within the regional model system itself. However, it is likely that a case of unequal domains for the regional atmosphere and ocean (the 'small regional ocean - big regional atmosphere' case!) applies, so some 2D global ocean/sea ice input is nevertheless required to the regional atmosphere.

In the framework of maximum modularity (cf. future studies), atmospheric, ocean, land surface and sea ice components should be considered separately even in regional applications, resulting in a requirement of specification of 2D input fields at the atmosphere-ocean, atmosphere-land surface, atmosphere-sea ice, and sea ice-ocean interfaces. As this is in principle analogous to the coupling between corresponding global model components, we refer to the respective sections in REDOC I.2, e.g. from the atmospheric, ocean, land surface and sea ice WPs. We also note that regional chemistry models might be used in the future as a part of the PRISM system and refer to the relations specified between AGCMs and AC-models.

For the global <-> regional coupling, we expect state variables to be passed (not fluxes) for the foreseeable future, motivated by the difference in spatial and temporal scales between global components and regional components. Coupling frequencies for lateral coupling can at best equal the time-step of the global model/dataset. This would be essential for two-way coupling. Currently, however, it seems adequate to do one-way coupling and couple with a longer interval (3 to 6 hours) and do time interpolation. This implies a corresponding delay in executing the regional component compared to the global one.

A future system should be able to deal with both options of (1) high coupling frequency and concurrent global and regional simulations and (2) lower coupling frequency, time-delayed global and regional simulations and time interpolation between coupling input.

*.REGIONAL ATMOSPHERE MODEL*

Field	Process	Unit	Freq.	Issues	Rating	Origin
Temperature	Atmosphere dynamics	K	Time step of A or up to 6 hours	3D in boundary zone	E	A
Humidity	Atmosphere dynamics	kg/kg	Time step of A or up to 6 hours	3D in boundary zone	E	A
Zonal velocity	Atmosphere dynamics	m/s	Time step of A or up to 6 hours	3D in boundary zone	E	A
Meridional velocity	Atmosphere dynamics	m/s	Time step of A or up to 6 hours	3D in boundary zone	E	A
Cloud water	Cloud/radiation	kg/kg	Time step of A or up to 6 hours	3D in boundary zone	D	A
Cloud ice	Cloud/radiation	kg/kg	Time step of A or up to 6 hours	3D in boundary zone	D	A
Cloud cover	Cloud/radiation	fraction	Time step of A or up to 6 hours	3D in boundary zone	D	A
Particles (sulfate cycle variables etc)	Radiation	various units, cf. atmosphere WP	Time step of A or up to 6 hours	3D in boundary zone	D	A / AC
Surface pressure	Atmosphere dynamics	Pa	Time step of A or up to 6 hours	2D in boundary zone	E	A
Land-Sea mask	Inter/extrapolation in boundary zone	-	Once per run	2D in boundary zone	E	A
Orography	Inter/extrapolation in boundary zone	m	Once per run	2D in boundary zone	E	A
Sea surface temperature	Ocean-atmosphere heat flux in case of non-identical domains for ARCM and ORCM or when there is no ORCM.	K	1h	2D for complete regional domain	E	O
Sea surface albedo	Reflection of solar radiation (possibly for different frequency bands) in case of non-identical domains for ARCM and ORCM or when there is no ORCM.	-	1h	2D for complete regional domain	M	O
Sea surface emissivity	Long-wave radiation in case of non-identical domains for ARCM and ORCM or when there is no ORCM.	-	1h	2D for complete regional domain	M	O
Sea surface roughness length	Turbulent fluxes in case of non-identical domains for ARCM and ORCM or when there is no ORCM.	m	1h	2D for complete regional domain	M	O
Sea ice fraction	Ocean/ice-atmosphere heat flux in case of non-identical domains for ARCM and ORCM or	fraction	1h	2D for complete regional domain	E	SI

Field	Process	Unit	Freq.	Issues	Rating	Origin
	when there is no ORCM.					
Sea ice surface temperature	Ice-atmosphere heat flux in case of non-identical domains for ARCM and ORCM or when there is no ORCM.	K	1h	2D for complete regional domain	E	SI
Sea ice albedo	Reflection of solar radiation (possibly for different frequency bands) in case of non-identical domains for ARCM and ORCM or when there is no ORCM.	-	1h	2D for complete regional domain	E	SI
Sea ice emissivity	Long-wave radiation in case of non-identical domains for ARCM and ORCM or when there is no ORCM.	-	1h	2D for complete regional domain	M	SI
Sea ice surface roughness	Turbulent fluxes in case of non-identical domains for ARCM and ORCM or when there is no ORCM.	m	1h	2D for complete regional domain	M	SI

### .REGIONAL OCEAN MODEL

Field	Process	Unit	Freq.	Issues	Rating	Origin
Potential temperature	ocean dynamics	K	1h	3D in ocean boundary zone	E	O
Salinity	Ocean dynamics	psu	1h	3D in ocean boundary zone	E	O
Zonal velocity	Ocean dynamics	m/s	1h	3D in ocean boundary zone	E	O
Meridional velocity	Ocean dynamics	m/s	1h	3D in ocean boundary zone	E	O
Sea surface height	Ocean dynamics	m	1h	2D in ocean boundary zone	E	O

### .REGIONAL SEA-ICE MODEL

Field	Process	Unit	Freq.	Issues	Rating	Origin
Sea ice concentration (frac. ice)	Ice dynamics	fraction	1h	2D in ocean boundary zone	E	SI
Sea ice thickness	Ice dynamics/thermodynamics	m	1h	2D in ocean boundary zone	E	SI
Sea ice velocity	Ice dynamics	m/s	1h	2D in ocean boundary zone	E	SI
Sea ice/snow surface temperature	Ice/snow thermodynamics	K	1h	3D in ocean boundary zone	E	SI
Thickness of snow on ice	Ice/snow thermodynamics	m	1h	3D in ocean boundary zone	E	SI
Distribution of ice classes	Ice dynamics	-	1h	3D in ocean boundary zone	M	SI
Sea ice temperature	Ice thermodynamics	K	1h	3D in ocean boundary zone	M	SI



**rating** = Essential, Desirable, Maybe, Unlikely in the next 5-10 years

## Requirements Summary Table

The table gives the summary of the responses obtained from the different model work packages to the questionnaire about the requirements on the coupler functionalities and the kind of simulation the PRISM system should be able to perform.

---

### General Requirements:

**I.4-1:** "The same version of a component model should be usable as part of a coupled model in the PRISM System and outside the PRISM system in stand-alone runs."

**I.4-2:** "The PRISM coupler allows intrinsic characteristics of the component models (e.g. length of a time step) to change at run-time".

**I.4-3:** "The PRISM coupler allows coupling data characteristics of the component models (e.g. units, grid co-ordinates, mask, parallel decomposition, ...) to change at run-time".

### Controller/Driver Requirements :

#### General:

**I.4-4:** "The PRISM System can also be used to assemble and run coupled models based on component models which do not conform to the PRISM Physical interfaces given that they include the well defined Model Coupling Technical Interface".

**I.4-5:** "The PRISM System can be used to assemble and run coupled models based on an arbitrary number of component models (not only the full coupled model assembling all PRISM model components)."

#### Model execution:

**I.4-6:** "The PRISM System should be able to run the different component models concurrently, in a regular sequence (one after the other), or in some pre-defined combination of these two modes."

**I.4-7:** "The PRISM System should be able to control *dynamic* model execution i.e. one or more component models may be launched and/or finish run-time at pre-determined points in the simulation."

**I.4-8:** "The PRISM System should be able to control *conditional* model execution i.e. one or more component models may be launched run-time during the simulation only if a particular scientific condition is met."

**I.4-9:** "The PRISM System should be able to control a global coupled system flexible in terms of executables (extreme are: each component is a separate executable -MPMD-, or all components run in parallel or in sequence within only one executable -SPMD)."

**I.4-10:** "The PRISM System should be able to give some statistic on the load balancing of the run."

#### Coupling exchanges management:

**I.4-11:** "End-point data exchange: when producing coupling data, the source model should not need to know what other model will consume it; when asking for coupling data a target model should not need to know what other model produces it."

Termination and restart:

**I.4-12:** "The PRISM System ensures that whole simulation shuts down cleanly (regular and unforeseen termination) in an intelligent way (e.g. after restart is saved) and report error if one component aborts."

**I.4-13:** "After a machine breakdown, the PRISM System ensures automatically a proper restart of the coupled system."

Other controls:

**I.4-14:** "The PRISM System warns the user if the coupling and I/O frequencies are not synchronised (in this case, an optimal load balancing leading to synchronisation of the different component models without idle time would be impossible to achieve)."

Transformer Requirements:

**I.4-15 to I.4-20:** "The PRISM coupler should provide the following transformations (OASIS transformation are given as examples):

**I.4-15:** Time operations:

- a):** time averaging
- b):** time interpolation
- c):** minimum or maximum over a certain time range
- d):** other time operations

**I.4-16:** 2D spatial interpolation:

- a):** nearest-neighbour (Ex: NNEIBOR)
- b):** nearest-neighbour gaussian weighted (Ex: GAUSSIAN)
- c):** bilinear (Ex: BILINEAR)
- d):** bicubic (Ex: BICUBIC)
- e):** 1st order conservative remapping (Ex: SURFMESH)
- f):** 2nd order conservative remapping
- g):** higher order conservative remapping
- h):** remapping using user-defined remapping info (e.g. runoff remapping) (Ex: MOZAIC)
- i):** other

**I.4-17:** 3D spatial interpolation:

- a):** nearest-neighbour
- b):** nearest-neighbour gaussian weighted

**c):** bilinear

**d):** bicubic

**e):** 1st order conservative remapping

**f):** 2nd order conservative remapping

**g):** higher order conservative remapping

**h):** remapping using user-defined remapping info

**i):** other

**I.4-18:** 1D spatial interpolation:

**a):** nearest-neighbour

**b):** nearest-neighbour gaussian weighted

**c):** bilinear

**d):** bicubic

**e):** 1st order conservative remapping

**f):** 2nd order conservative remapping

**g):** higher order conservative remapping

**h):** remapping using user-defined remapping info

**i):** other

**I.4-19:** Other transformations:

**a):** Conservation: ensure global energy conservation between source and target grid (Ex: CONSERV)

**b):** Combination: of different parts of different coupling fields or of other predefined external data (Ex: FILLING)

**c):** Algebraic operations: with possibly different coupling fields or predefined external data and numbers as operands (Ex: BLASOLD, BLASNEW, SUBGRID, CORRECT)

**d):** Specific algebraic transformations

**d1):** Celsius <-> Kelvin

**d2):** Degree <-> Radian

**e):** Indexing operations:

**e1:** Mask: Only the points listed in index have meaningful data and the others are changed to missing (Ex: MASK)

**e2:** Scatter: scatters the model data onto the points listed in index (1st or 2nd order spatial extrapolation) (Ex: EXTRAP)

**e3:** Gather: gathers from the input data all the points listed in index

**f):** Spatial "collapse" operations: collapse of any dimension or combination of dimensions by various -possibly weighted- statistical operations (mean, max, min, etc.)

**g):** Subspace: extraction of subspaces or hyperslabs in any combination of spatiotemporal or other dimension

**h):** Merge: replace n dimensions by an index dimension by sampling at n-dimensional points, e.g. replace a lat-lon-height field with value along a trajectory

**I.4-20:** *Others* - please specify.

**I.4-21:** "The PRISM coupler should be able to give some statistics on the coupling fields (mean, max, min, etc.)

Ex: CHECKIN, CHECKOUT in OASIS

**I.4-22:** "The PRISM coupler should support source and target coupling domains that totally or partially overlap (e.g. global atmosphere with a regional ocean, regional model nested into global model, etc.)"

**I.4-23:** "The PRISM coupler should automatically perform some basic transformations (units -e.g. Celsius to Kelvin, order of dimensions -e.g. source (x,y,z) - target (z,y,x), etc.)"

**I.4-24:** "The PRISM coupler should recognise type of coupling data (flux, vector, scalar) based on meta-data description, and automatically provides relevant transformation operations (this choice of transformation will be validated or invalidated by the user)."

Lead	Atmosphere			Atmospheric chemistry			Land surfaces			Ocean			Sea-ice			Ocean biogeochem.			Regional Models		
	S. Planton			G. Brasseur			J. Polcher			E. Guilyardi			H. Drange			C. Le Quéré			M. Rummukainen		
	E	D	MUN	E	D	MUN	E	D	MUN	E	D	MUN	E	D	MUN	E	D	MUN	E	D	MUN
I.3-1	X				X		See specific remarks below			X				X		X			X		
I.3-2		X		X								X			X			X			X
I.3-3		X		X			X				X						X				X
I.3-4			X			X				X				X		X					X
I.3-5	X				X					X			X			X			X		
I.3-6	X				X					X			X			X			X		
I.3-7	X			X						X				X		X					
I.3-8	X				X						X			X		X				X	
I.3-9	X			X						X			X			X			X		
I.3-10		X		X						X			X	X		X			X		
I.3-11			X	X						X			X				X		X		
I.3-12	X			X						X			X			X			X		
I.3-13	X			X						X			X			X			X		
I.3-14		X		X						X			X	X		X			X		
I.3-15a		X		X						X			X			X			X		
I.3-15b	X			X						X			X			X			X		
I.3-15c		X			X					X				X		X				X	
I.3-15d		X			X									X			X				X
I.3-16a	X				X					X			X	X		X			X		
I.3-16b	X					X				X				X		X			X		
I.3-16c	X			X						X			X			X			X		
I.3-16d	X				X					X			X			X			X		
I.3-16e	X			X						X			X			X			X		
I.3-16f	X					X				X			X			X			X		
I.3-16g	X					X				X			X				X		X		
I.3-16h	X				X					X			X				X		X		
I.3-16i			X	X										X					X		
I.3-17a	X			X						X				X		X			X		
I.3-17b	X				X					X				X		X			X		
I.3-17c	X			X						X			X			X			X		
I.3-17d	X				X					X			X			X			X		
I.3-17e	X			X						X			X			X			X		
I.3-17f	X					X				X			X			X			X		
I.3-17g	X					X					X						X		X		
I.3-17h	X				X					X			X				X		X		
I.3-17i			X	X										X					X		
I.3-18a	X			X						X				X		X					X
I.3-18b	X				X					X				X		X					X
I.3-18c	X			X						X			X			X					X
I.3-18d	X				X					X			X			X					X
I.3-18e	X			X						X			X			X					X
I.3-18f	X					X				X			X			X					X
I.3-18g	X					X				X			X				X				X

I.3-18h	X				X				X			X			X			X		
I.3-18i			X		X				X		X				X			X		
I.3-19a	X			X					X			X			X			X		
I.3-19b	X	X		X					X			X			X			X		
I.3-19c	X			X					X			X			X			X		
I.3-19d1	X			X				X	X			X			X			X		
I.3-19d2	X			X				X	X			X			X			X		
I.3-19e1	X			X				X	X			X			X			X		
I.3-19e2	X			X				X	X			X			X			X		
I.3-19e3	X			X				X	X			X			X			X		
I.3-19f		X			X			X	X			X			X			X		
I.3-19g		X		X				X	X			X			X			X		
I.3-19h		X		X				X	X			X			X			X		
I.3-20		X		X							X							X		
I.3-21	X			X				X			X			X				X		
I.3-22	X				X			X			X			X				X		
I.3-23		X		X				X			X			X				X		
I.3-24		X			X			X			X			X				X		
	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>	<b>EDMUN</b>		
	<b>Atmosphere</b>		<b>Atmospheric chemistry</b>			<b>Land surfaces</b>			<b>Ocean</b>			<b>Sea-ice</b>			<b>Ocean biogeochem.</b>			<b>Regional Models</b>		

## Acknowledgements

We acknowledge the fruitful inputs from Gervan Madec (IPSL/LODYC, Paris) and Olivier Marti (IPSL/LSCE, Saclay) at early stages in the discussion.

## Coupling Fields

Code	Field name	Units	Definition & Conventions
<b>1: atmosphere to Ocean-surface module exchange</b>			
1.1	Rainfall	kg/m <sup>2</sup> /s	Mass flux, positive downwards, includes all liquid precipitation
1.2	Snowfall	kg/m <sup>2</sup> /s	Mass flux, positive downwards, includes all solid precipitation
1.3	Incoming solar radiation	W/m <sup>2</sup>	Energy flux, positive downwards
1.4	Solar zenith angle	radians	
1.5	Fraction of diffuse solar radiation	W/m <sup>2</sup>	Energy flux, positive downwards
1.6	Downward infrared radiation	W/m <sup>2</sup>	Positive downward
1.7	Sensitivity of atmos. T and q to surface fluxes		$\delta T/\delta Q_s$ and $\delta q/\delta Q_s$
<b>2: ocean-surface module to atmosphere exchange</b>			
2.1	Sensible heat flux	W/m <sup>2</sup>	Energy flux, positive upwards
2.2	Latent heat flux	W/m <sup>2</sup>	Energy flux, positive upwards
2.3	Surface emissivity		
2.4	Albedo, direct	-	
2.5	Albedo, diffuse	-	



2.6	Surface radiative temperature	K	
2.7	Evaporation	kg/m <sup>2</sup> /s	Mass flux, positive upwards
2.8	Wind stress	N/m <sup>2</sup>	Momentum flux, vector
<b>3: atmosphere to surface layer turbulence exchange</b>			
3.1	Mean sea level surface pressure	hPa	
3.2	Air temperature at lowest level	K	
3.3	Air humidity at lowest level	g/g	
3.4	Wind at lowest level	m/s	Vector
3.5	Wind module at lowest level	m/s	Possibly including gustiness effects
3.6	Lowest level height	m	
<b>4: surface layer turbulence to ocean-surface module exchange</b>			
4.1	$\rho C_d$ drag coefficient	kg/m <sup>2</sup> /s	Surface layer exchange coefficient for momentum
4.2	$\rho C_e$ exch. coeff.	kg/m <sup>2</sup> /s	Surface layer exchange coefficient for sensible heat
4.3	$\rho C_h$ exch. coeff.	kg/m <sup>2</sup> /s	Surface layer exchange coefficient for moisture
<b>5: ocean-surface module to surface layer turbulence exchange</b>			
5.1	Surface temperature	K	
5.2	Surface roughness		
5.3	Displacement height		

Code	Field name	Units	Definition & Conventions
<b>6: ocean-surface module to ocean exchange</b>			
6.1	Non solar heat flux	W/m <sup>2</sup>	Energy flux, positive upwards
6.2	Solar radiation	W/m <sup>2</sup>	Energy flux, positive downwards
6.3	Fresh water flux	kg/m <sup>2</sup> /s	Mass flux, positive downwards
6.4	Salt flux	kg/m <sup>2</sup> /s	Mass flux, positive downwards
6.5	Wind stress	N/m <sup>2</sup>	Momentum flux, vector
6.6	Wind work	(m/s) <sup>3</sup>	U <sup>3</sup>
6.7	Mass of snow and ice	kg	
<b>7: ocean to ocean-surface module exchange</b>			
7.1	Temperature at sea-ice base	C	
7.2	Sea surface temperature	C	
7.3	Surface radiative temperature	C	
7.4	Surface current	m/s	Vector
7.5	Sea surface salinity	PSU	
7.6	Sea surface height	m	
7.7	Absorbed solar radiation in first oceanic layer	W/m <sup>2</sup>	
<b>8: land surface scheme to ocean exchange</b>			
8.1	Continental runoff	m <sup>3</sup> /s	Volume flux, positive towards ocean

## *Tables*

Table 1: Examples of Universal Parameters .....	20
Table 2: Links to the input required for the standard physical and algorithmic interface .....	22
Table 3: Features of the GUI.....	35
Table 4: Job Control: Functions and the Graphical User Interface .....	38
Table 5: Feature and Capability Listing.....	39
Table 6: Results and the GUI.....	39
Table 7: Access and the GUI .....	40
Table 8: Security – terms and the GUI .....	40
Table 9: Authentication and the GUI .....	40
Table 10: Security level of operations .....	41
Table 11: PRISM actors and main activities.....	47
Table 12: Table of different processes .....	48
Table 13: Risks.....	57
Table 14: Graphics packages considered for PRISM.....	79
Table 15: Examples of universal parameters .....	104
Table 16: Software components in the system.....	115
Table 17: PRISM system messages .....	117
Table 18: Case 1, administration and maintenance advantages .....	118
Table 19 Case 2, configurability of the .....	119
Table 20: Case 3, monitoring and control of anexperiment .....	120
Table 21: Software components and implementations.....	122
Table 22: Software components and their location.....	128
Table 23: Risks.....	134
Table 24: Transformations on 2D scalar coupling fields .....	148
Table 25: Input from Hadley Centre, WP3f-3h, CERFACS, ECMWF, KNMI, MPI, Météo France	157
Table 26: Required Functionality for the PRISM graphics package.....	159



## *Figures*

Figure 1: Xcdp graphical view of an experiment,	44
Figure 2: PRISM user interactions	46
Figure 3: Central Site Architecture, directory centric	49
Figure 4: Common Data Architecture, model provider centric	50
Figure 5: Full replication architecture, no central repository	50
Figure 6: PRISM software deployment model	51
Figure 7: two models are <b>sequential by nature</b>	60
Figure 8: Two models are <b>concurrent by</b>	61
Figure 9: Two models are <b>concurrent by</b>	61
Figure 10: Two models are <b>sequential by construction</b>	62
Figure 11: Two per two approach	69
Figure 13: Diagram of the archive, Data Processing and Visualisation System	80
Figure 14: Internal Data Management structure for the Job Flow an Run Shell	81
Figure 15: TOP 500-Architecture evolution over years.	91
Figure 16: A proposal for standard interfaces	107
Figure 17: PRISM user interacting with the central and local PRISM sites	111
Figure 18: Configuration process in PRISM:	112
Figure 19: Service lookup in WS	114
<i>Figure 20: WS subsystem composition</i>	115
Figure 21 :System processes in PRISM	116
Figure 22: Case 1, User downloads GUI and directory info:	118
Figure 23: Case 2: User submits experiment;	119
Figure 24: Case 3, User monitors experiment -	120
Figure 25: A central site and a PRISM site with all components in place.	121
Figure 26: The PrepIFS system and its software components.	123
Figure 27: Monitoring and scheduling of experiments;	124

Figure 28: PRISM subsystem software components	127
Figure 29: Details of the different parts of the coupled PRISM model	136
Figure 30: Diagram of the Archive, Data Processing, and Visualization system	155
Figure 32: High-end graphics	158

Version: Handbook1.0.3.RB

Date: March 7 2003