SALOME Demonstration
SALOME

Downloading and installing
Get the open-source version

Public web site: http://www.salome-platform.org
Installing the open-source version

Standard installation wizard
SALOME

Integrating a computation code
Summary

- Integration into SALOME platform
- Programming model
- Tools
  - MED
  - XDATA
  - HXX2SALOME
  - YACSGEN
- Some integration examples
  - SALOME-MECA
  - PARAD
  - CARMEL
Integration into SALOME platform
Integration into SALOME platform: Global view of a SALOME platform

- **User Interface**
  - GUI
  - TUI

- **Communication layer**
  - Services
  - Salome Engines
    - Component
    - Component
    - MyCompo

- **Services**
  - MyGUI
  - MyTUI

- **Integration into SALOME platform**
  - Global view of a SALOME platform
Integration into SALOME platform: GUI only integration

User Interface
- GUI
- TUI
- MyGUI

Services

Communication layer

Salome Engines
- Component
- Component

Services
Integration into SALOME platform:
Component only integration

- User Interface
  - GUI
  - TUI
  - MyTUI
- Communication layer
- Salome Engines
  - Component
  - Component
  - MyCompo
- Services

User
SALOME Programming model
SALOME Programming model: Implementing a service

- Each service is implemented by a SALOME component
- SALOME component programming model is composed of two layers:
  - SALOME objects
  - DSC component model (Dynamic Software Component)
SALOME programming model: main principles

- SALOMES is based on CORBA
  - Used to handle communication between SALOME modules

- CORBA benefits
  - Object distributed model
  - Mapping on different languages: C++ and python used
  - Handle machine heterogeneity

- Parallel distributed model in progress (with INRIA)
  - Enable parallel communications between parallel CORBA objects
    - Named PaCO++
SALOME Programming model: SALOME objects

- SALOME object extends CORBA object
  - Add a new interface to handle distributed object life cycle
  - Add a generic server of SALOME object named SALOME container
SALOME Programming model: SALOME object and YACS service

- A YACS Service is a method of a SALOME object described into a component XML file
  - Dataflow ports are the input and the output arguments of the method
    - Ex: void run(in string data, out string result)
- All the modules of the platform use this model
  - Ex: Mesh, CAD modeling
  - They define services that can be used into a YACS schema
SALOME Programming model: DSC - Dynamic Software Component

- History: SUPERV tool (old SALOME code coupling system)
  - Only Workflow and Dataflow
  - Use an external coupling tool for Datastream: CALCIUM®
    - It’s a supervisor too….
    - Fixed types

- DSC objectives
  - Datastream communications into YACS
  - User types like structures
  - Interface ports for RMI like connection
SALOME Programming model: DSC - Dynamic Software Component

- An extension to SALOME object concept
  - Add uses and provides port to SALOME object

- A port is a CORBA interface
  - Provides port: implement the interface
  - Uses port: connected to a provides port to use the interface
  - Each port can have properties

- CORBA IDL is not modified
  - Ports are added at runtime
  - They are described into an XML file
    - Used by YACS

Diagram:
- DSC Component
- CORBA Interface
- SALOME object provides uses
SALOME Programming model: DSC - Dynamic Software Component

- **DSC Basic layer**
  - Dynamic declaration of ports
  - Connection between ports
    - A uses port can be connected to many provides ports (uses multiple in CCM)
    - A provides port can be connected to many uses ports
    - Each port is notified when his connection state changes

Diagram:
- 1 to N
- N to 1
SALOME Programming model: DSC - Dynamic Software Component

- **DSC User layer**
  - Provides data oriented ports (put/get operations)
    - Implement different policies from data-oriented code coupling tools
      - as former EDF coupling supervisor named CALCIUM®
      - Ex: put(data, time)

- **Tools to help the implementation/integration of codes**
  - Generates wrappers for FORTRAN 77, Python and C++
  - Uses ports defined in DSC User layer
    - Named YACSGen
SALOME Programming model:

Summary

- A service is a method of a SALOME component (DSC + SALOME object)
  - Implicit Workflow port
    - Managed by YACS
  - Dataflow ports are the input and the output arguments of the method
    - Ex: void run(in long data, out long result)
  - Datastream ports are the DSC ports
  - Dataflow and Datastream ports are described into an XML file
Tools - MED
A data model: MED provides a model for storing and recovering data associated to numerical meshes and fields.

A library of structures to hold data on meshes and fields, offer persistency and facilitate the exchange between codes.

A three level interface MED-file, MED-memory and MED-CORBA.

A toolkit of services: localisation, interpolation, arithmetic operations, norms, parallelism.
MED defines a model for storing and recovering data associated to numerical meshes and fields

- Structured and unstructured meshes
- Fields
  - On cells, faces, vertexes
  - Gauss points
- Groups
- Descending and nodal connectivity
- Types: TRIA3, TRIA6, QUAD4, QUAD8, POLYGON, TETRA4, TETRA10, HEXA8, HEXA20, PENTA6, PENTA16, POLYHEDRA
Implement the MED model in a library that allows to read and write meshes & fields

- Above HDF5
- C / FORTRAN API
- EDF Project (outside SALOME)
Tools - MED : MED-memory

- C++ API that allows creating mesh and field objects in memory
- Drivers: MED / VTK / ENSIGHT / CASTEM / PORFLOW
- Localisation, projection, interpolation, arithmetic operations
- Code coupling by exchanging data on meshes and fields with MPI or CORBA
Tools - MED : Interpolation Component (Principle)

Mesh (i,j,k)

NK Meshing in MED (CRONOS2)

TH Meshing in MED (FLICA4)

Mesh (I,J,K)

Geometry Interpolation Component (INTERP_2_5DCPP)

Mesh-to-mesh volume intersection:

Code-to-code data transfer $f <> F$:
(Temperatures, Mod. Density, etc.)

$$\Omega_{I,J,K}(i,j,k) = \sum_{i,j,k} f(i,j,k) \cdot \Omega_{I,J,K}(i,j,k)$$

$$F(I,J,K) = \frac{\sum_{i,j,k} \Omega_{I,J,K}(i,j,k)}{\sum_{i,j,k} \Omega_{I,J,K}(i,j,k)}$$
Tools - MED: Towards parallel computing

- Parallel MED file: storage for meshes and fields
- Parallel meshing algorithm
- Med splitter
- Parallel mesh manager
- Parallel interpolation
- Parallel supervisor
- Code A
- Code B
- Data reduction
- Parallel visu
How to define specific data in SALOME

- SALOME provides an exchange model for main data in simulation: CAD, meshes, fields (MED).
- Solvers often require other specific data: material, boundary conditions, numerical settings…
- We cannot define a common data model that covers every field.
- Need to provide an interface in a script command language and a GUI interface for those data.
Tools – XDATA : Principle

- A generic tool to define specific data model that can be integrated in SALOME GUI and TUI
  - Definition of data model with control
  - Use Python language
  - Automatic generation of GUI from data model
  - Automatic SALOME integration: study management, supervisor
uox = Material( name='UOX 4%', density=10.85,
               color='green')

zrc = Material( name='Zircaloy-4', density=6.55,
                color='gray')

helium = Material( name='Helium', color='lightblue')

pellet = Pellet(  name='UOX 4%'
                , material= uox
                , internal_radius = 0.
                , external_radius = 0.004095
                , height        = 0.0115
                , void_ratio    = 0.05
               )
# This class shows how to define a Material object
# A Material has three properties:
# a name
# a density (default = 1.)
# a color

class Material( XObject ):
    __init_xattributes__ = [
        XAttribute( 'name', xtype=XString() ),
        XAttribute('density', xtype=XFloat(min=0), default_value=1.),
        XAttribute( 'color', xtype=XString(), default_value='')
    ]
Dialog box automatically generated from data model
Tools - XDATA : Example

- DYN3D integration into SALOME platform: development by FZD of a graphical pre-and-post processing tool.
Tools - XDATA : Example

- Neutronics – Thermalhydraulics coupling : MSLB data model
Tools – Hxx2salome
A tool that takes charge of the SALOME wrapping:

- Entry: a C++ standalone component
- Generates a SALOME component that wrap C++ component

Diagram:

- C++
- CPP
- C, C++, Fortran legacy applications (libraries)
- Automatic generation
- SALOME/CORBA
- SALOME

EDF R&D
Tools – Hxx2salome : C++ integration : Presentation

- Hxx2salome - a tool that manages:
  - The code generation
  - The module component compilation
  - The update of environment
  - Textual and graphical interface
Tools – Hxx2salome : C++ integration : Presentation

Principles:

- **C++ interface**
  - Definition of a class in a header

- **SALOME interface**
  - An IDL interface containing all the compatible methods

- **SALOME implementation**
  - Generated mapping of the IDL interface using C++

- **Component catalog**

  ![Diagram](image)

  - Extraction of functions
  - Type analysis: selection of compatible methods
  - Using type information
  - Using Salome tool runIDLParser
### Tools – Hxx2salome: Supported types

<table>
<thead>
<tr>
<th>C++ Argument type</th>
<th>IDL associated type</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>in long</td>
</tr>
<tr>
<td>double</td>
<td>in double</td>
</tr>
<tr>
<td>float</td>
<td>in float</td>
</tr>
<tr>
<td>short</td>
<td>in short</td>
</tr>
<tr>
<td>const char*</td>
<td>in string</td>
</tr>
<tr>
<td>const std::string&amp;</td>
<td>in string</td>
</tr>
<tr>
<td>int&amp;</td>
<td>out long</td>
</tr>
<tr>
<td>double&amp;</td>
<td>out double</td>
</tr>
<tr>
<td>std::string&amp;</td>
<td>out string</td>
</tr>
<tr>
<td>const MEDMEM::MESH&amp;</td>
<td>in SALOME_MED::MESH</td>
</tr>
<tr>
<td>const MEDMEM::MESH*</td>
<td>in SALOME_MED::MESH</td>
</tr>
<tr>
<td>const MEDMEM::FIELD&lt;double&gt;*</td>
<td>in SALOME_MED::FIELDDOUBLE</td>
</tr>
<tr>
<td>const MEDMEM::FIELD&lt;double&gt;&amp;</td>
<td>in SALOME_MED::FIELDDOUBLE</td>
</tr>
<tr>
<td>const std::vector&lt;double&gt;&amp;</td>
<td>in SALOME::Sender</td>
</tr>
<tr>
<td>MEDMEM::FIELD&lt;double&gt;*&amp;</td>
<td>out SALOME_MED::FIELDDOUBLE</td>
</tr>
</tbody>
</table>
Tools – Hxx2salome : Example - Generic API for core neutronics and thermalhydraulics

class FLICA
{
  public:
    void start(long gematRam, long debug, const char *outputPath,
    const char *useCasePath, const char *useCaseFile,
    const char *geometry, const char *boundaries,
    const char *power, const char *postProcess);
    void end();
    void eval(const char *command);
    void init();
    void reinit();
    long steadyState(long nMaxIterations);
    long transient(double maxTimeStep);
    MEDMEM::MESH *getMedMesh(const char *meshname);
    MEDMEM::FIELD<double> *getMedField(const char *fieldName);
    void coupleGeometry(const std::vector<double>& coordx,
    const std::vector<double>& coordy, const std::vector<double>& coordz,
    const char *geometryType);
    MEDMEM::FIELD<double>* getFeedback(const char *parameterName);
    void setPower(const MEDMEM::FIELD<double>& Wnorm, double WtotFuel,
    double WtotFluid);
    void saveInFile(const char *fileName);
};
Tools - YACSGEN
Tools – YACSGEN : Presentation

- A wrapping generator for C++, Fortran or Python existing code with emphasis on datastream coupling
  - Entry: existing code(s) and a description of expected SALOME component(s)
  - Generate a SALOME module that wraps the existing code
  - Handle datastream ports unlike Hxx2salome

YACSGEN

Module and components description

C, C++, Fortran legacy applications (libraries)

SALOME application

SALOME module

+ other existing SALOME modules
Tools - YACSGEN : Module description

- Module : name
  - 1 or more Component : name, implementation (F77, Python, C++)
    - 1 or more Service : name
      - Dataflow ports (in, out) : name, type
      - Datastream ports (in, out) : name, type, mode
      - Body : source code
    - Added sources or libraries
  - Description language : Python with some classes added (Module, Component, Service)
  - Possible to define and generate a full installed SALOME application
Use 3 existing codes to solve a heat transfer problem:

- FLUID code to calculate temperature in fluid flow with heat transfer boundary condition
- SOLID code to calculate temperature in a solid by solving a diffusion equation with heat transfer boundary condition
- NEUTRO code to calculate the heat source term in solid
Tools – YACSGEN: Hello world example: coupling diagram

FLUID calculation
\[ t \rightarrow t + dt \]
Projection on solid boundary

SOLID calculation
\[ t \rightarrow t + dt \]

NEUTRO calculation
\[ t \rightarrow t + dt \]

Boundary temperature \((t+dt)\)
Fluid heat transfer coefficient \((t+dt)\)

Heat source \((t)\)

Boundary temperature \((t)\)
Solid heat transfer coefficient \((t)\)

Projection on fluid boundary
Tools – YACSGEN: First step: add datastream ports to codes

- **Datastream port**
  - Name, Type (R, I, S, B, C), Direction (IN, OUT)
  - Datas are tagged: Mode (TIME, ITERATION)

```
Program Thermix
...
C Beginning of time loop
C import variable Power to process time step (ti, tf)
   CALL cplre(compo,CP_TEMPS,ti,tf,i,"Power",nmax,nval,P,info)
...
C process time step ti,tf
...
C export computed variable Temperature at time tf
   CALL cpere(compo,CP_TEMPS,tf,i,"Temperature",nval,temp,info)
C end of time loop
...
```
Thermix code requests $Power(ti)$ that is produced by Neutronix code ($PowNeutro$)

$$Power(ti) = a \times PowNeutro(t+dt) + (1-a) \times PowNeutro(t)$$
If we want to define one component

**YACSGEN definition**

```python
c1=F77Component("FLUID",
    services=[Service("run",
        inport=[("dt","double")],
        instream=[("Tsolid","CALCIUM_double","T")],
        outstream=[("Temperature","CALCIUM_double","T")],
    ),
    libs="-lfluid")
```

The existing FLUID library

**Module generation : call generate and it’s done**
Tools – YACSGEN : YACS final diagram
Some integrations examples: Three typical SALOMÉ applications

- **SALOMÉ-MECA Application**
  - What? Platform designed for implementing a complete FEM simulation process in the domain of solid structure analysis
  - How? Integration of SALOMÉ and Code_Aster software environments through the SALOMÉ ÅSTER module.

- **CARMEL Application**
  - What? Demonstrator designed to validate a multi physical modelization (thermal – magnetic experiment)

- **PARAD Application**
  - What? Domain specific tool designed for the radio-protection engineering process
Some integrations examples: SALOME-MECA Application (1/2)

- The main features (in addition to classical SALOME features):
  - Modelization of boundary conditions (module EFICAS)
  - Modelization of a complete physical problem (module ASTER)
  - Wizard to ease the creation of a new physical problem (typical use cases)
  - Use SALOME services from Code_Aster environment

CAD, Mesh

Physical Problem

Post-processing
Some integrations examples: SALOME-MECA Application (2/2)

The integration architecture

[Diagram showing the integration architecture with modules and components connected by arrows indicating use, drive, embed, and communication layer.]
Some integrations examples CARMEL : Application (1/2)

- The main features (in addition to classical SALOME features):
  - Magnetic modelization using SALOME CARMEL Module (generated with XDATA)
  - Thermal modelization using SALOME-MECA (Code_Aster engine)
  - Code coupling using SALOME YACS Module
Some integrations examples: CARMEL Application (2/2)

The integration architecture:

ASTER Modules

ASTER GUI

ASTER Components

YACS Module

CARMEL Module

CARMEL GUI

CARMEL Components

Code_Carmel solver engine

XDATA

input

generate