An overview of the DANSE software architecture

Michael Aivazis
Caltech

DANSE Kick-Off Meeting
Pasadena
Aug 15, 2006
User stereotypes

- Visiting scientist
  - *occasional user of prepackaged and specialized analysis tools*

- Instrument specialist
  - *author of prepackaged specialized tools*

- Expert scientist
  - *prospective author/reviewer of PRL paper*

- Analysis expert
  - *author of analysis, modeling or simulation software*

- Software integrator
  - *responsible for extending software with new technology*

- Framework maintainer
  - *responsible for maintaining and extending the DANSE infrastructure*
Distributed services
Pyre: the integration architecture

Pyre is a **software architecture**:  
- a specification of the organization of the software system  
- a description of the crucial structural elements and their interfaces  
- a specification for the possible collaborations of these elements  
- a strategy for the composition of structural and behavioral elements

Pyre is multi-layered  
- flexibility  
- complexity management  
- robustness under evolutionary pressures

Pyre is a **component framework**
Overview

Projects
- Caltech ASC Center (DOE)
- Geodynamics (NSF):
  - GeoFramework: $3M ITR, currently in its last year
  - CIG (www.geodynamics.org): $6.67M over 5 years
- Radar interferometry (ROI_PAC)
  - NASA and JPL internal funds

Portability:
- languages: C, C++, F77, F90
- compilers: all native compilers on supported platforms, gcc, Absoft, PGI
- platforms: all common Unix variants, OSX, Windows

Statistics:
- 1200 classes, 75,000 lines of Python, 30,000 lines of C++
- Largest run: nirvana at LANL, 1764 processors for 24 hrs, generated 1.5 Tb
The integration framework is a set of co-operating abstract services.
Writing python bindings

Given a “low level” routine, such as

```cpp
double danse::stableTimeStep(const char *);
```

and a wrapper

```cpp
char pydanse_stableTimestep__name__[] = "stableTimestep";
PyObject * pydanse_stableTimestep(PyObject *, PyObject * args)
{
    double dt = danse::stableTimeStep("deformation");

    return Py_BuildValue("d", dt);
}
```

• one can place the result of the routine in a python variable

```python
dt = danse.stableTimestep()
```

• The general case is not much more complicated than this
Encapsulating critical technologies

- **Extensibility**
  - *new algorithms and analysis engines*
  - *technologies and infrastructure*

- **High end**
  - *visualization*
  - *easy access to large data sets*
    - *single runs, backgrounds, archived data*
    - *metadata*
  - *distributed computing*
  - *parallel computing*

- **Flexibility:**
  - *interactivity: web, GUI, scripts*
  - *must be able to do almost everything on a laptop*
Quality assessment

- Correctness
- Logical coherence
- Robustness
  - core dumps in the analysis codes
    - prevent, handle
  - environment errors
    - outages
    - misconfigurations
- Usability
  - making everybody happy…
- Aesthetics
Maintenance and support

- Source control
- Robust and portable software build procedure
- Modern software design and implementation practices
  - documentation of requirements, use cases, design, algorithms
  - agile programming: “test before you implement”
- Quality control automation:
  - regular (nightly) builds of the software base
  - unit testing
  - regression testing
- Tracking of bug and feature requests
- Release management
- Strong community involvement
  - electronic notebooks, wiki, …
Support for concurrent applications

- Python as the driver for concurrent applications that
  - are embarrassingly parallel
  - have custom communication strategies
    - sockets, ICE, shared memory

- Excellent support for MPI
  - `mpipython.exe`: MPI enabled interpreter (needed only on some platforms)
  - `mpi`: package with python bindings for MPI
    - support for staging and launching
    - communicator and processor group manipulation
    - support for exchanging python objects among processors
  - `mpi.Application`: support for launching and staging MPI applications
    - descendant of `pyre.application.Application`
    - auto-detection of parallelism
    - fully configurable at runtime
    - used as a base class for user defined application classes
Support for distributed computing

- We are in the process of migrating the existing support for distributed processing into **gs1**, a new package that completely encapsulates the middleware.

- Provide both user space and grid-enabled solution

  **User space:**
  - *ssh*, *scp*
  - *pyre service factories and component management*

  **Web services**
  - *pyglobus*

- **Advanced features**
  - *dynamic discovery for optimized deployment*
  - *reservation system for computational resources*
Status

- Leveraging **pyre**:
  - large number of existing services
  - component cores
    - integration strategy well understood
  - component harnesses
    - support for user configurable properties is complete
  - data transport

- Under construction
  - user interfaces
    - two application-generic prototypes under construction
  - engines for analysis, simulation and modeling
  - application-specific UI
  - tests, of all sorts…
  - documentation
  - training materials