We have developed LOOS (Lightweight Object Oriented Structure-analysis) as a tool for making new tools to analyze molecular simulations. LOOS is an object-oriented library designed to facilitate the rapid development of new methods for structural analysis. LOOS includes over 130 pre-built tools for common structural analysis tasks. LOOS supports reading the native file formats of most common simulation packages and can write NAMD formats (PDB and DCD). A dynamic atom selection language, based on the C expression syntax, is included as part of the library and is easily accessible to both the programmer and the end user. LOOS is written in C++ and makes extensive use of Boost and the Standard Template Library. Through modern C++ design, LOOS is both simple to use (requiring knowledge of only 4 core classes and a few utility functions) and easily extensible. A Python interface to the core components of LOOS is also available, further facilitating rapid development of analysis tools and broadening the LOOS community by making it accessible to those who would otherwise be deterred by using C++. LOOS also includes a set of libraries and tools for performing elastic network model calculations that are easily extended to accommodate new methods.

**Selection Language**

- Based upon C/C++ expressions
- Built using lex & yacc
- Available via function call for all tools
- Select atoms via atom metadata
- Keywords bound to atom properties
  - id, name, resname, resid, name
  - Special keywords
    - all, none, hydrogen

**Select non-hydrogen atoms**
hydrogen

**Select CA atoms**

name == "CA"

**Select backbone atoms**

name =~ "^([C|O])([A|C])$"

**Select heavy atoms from a range of residues**

(resid) >= 10 && resid <= 20 && hydrogen

**Developing with LOOS**

- Flat hierarchy
- AtomicGroup can be a residue, chain, molecule, system, …
- Bulitin functions recover hierarchy:
  - splitByMolecule()
  - splitByAtom()…
- Selection is a copy (shared atoms)
- Number extraction operator
  - for loop to process trajectory
- Use factory functions to read models and trajectories:
  - Write out a PDB by printing it

**Examples**

```python
for frame in ptraj:
    frames_to_skip = int(sys.argv[5])
    traj_file = sys.argv[2]
    import sys
    print ptraj.currentIndex(), distance, angle, tors
```

- Integrate PROPKA with LOOS tools:
- Apply alignment transformation to desired subset of atoms
- Align desired subset of atoms
- Apply alignment transformation to frame (or subset)

**Example Code**

- Core library available in Python
- Support for shallow and deep copies
- Container classes are iterable
- STL containers explicitly wrapped
- Use Numpy rather than LOOS for linear algebra libraries
- ByTraj wraps a Trajectory
  - Use for-loop to process trajectory
  - Skip the first frames
  - Sride through trajectory
- PyLibTrapez wraps optimally aligned virtual trajectory
- Haversine alignment procedure
- Align desired subset of atoms
- Apply alignment transformation to frame (or subset)

**Benchmarks**

- Sensitivity Align Structures
  - C++
  - Python
- All-to-all RMSD
  - C++
  - Python
- Inter-atomic Distance
  - C++
  - Python
- Trajectory Size (MB)
  - C++
  - Python
- System Size (Molecules)
  - C++
  - Python
- Number of Frames
  - C++
  - Python
- Selection Size (Atoms)
  - C++
  - Python

**Class Structures**

- Lightweight
  - Tool developers only need 4 core classes:
    - Coord, Atom, AtomicGroup
  - Few external dependencies:
    - Boost, scons, atlas/lapack
  - All available via package system on Linux
- Extensible
  - Poly morphic classes
  - Algorithm encapsulation
  - Design patterns for easy extension
- Powerful
  - Null atom selection language
  - Parser built using standard Unix tools
  - Many useful member functions
  - SharedAtoms via Boost shared pointers
  - Simplifies memory management
- Copies are lightweight
  - Standard Template Library support
  - Support for basic periodicity
- Easy to use
  - Complex tools with minimal code
  - Python interface to core library
  - Rapid development of new tools
  - Templates for writing new tools for common case
  - Tools are self-documenting
- Consistent command line options across tools

**Example Code**

```python
for frame in ptraj:
    frames_to_skip = int(sys.argv[5])
    traj_file = sys.argv[2]
    import sys
    print ptraj.currentIndex(), distance, angle, tors
```

- Core library available in Python
- Support for shallow and deep copies
- Container classes are iterable
- STL containers explicitly wrapped
- Use Numpy rather than LOOS for linear algebra libraries
- ByTraj wraps a Trajectory
  - Use for-loop to process trajectory
  - Skip the first frames
  - Sride through trajectory
- PyLibTrapez wraps optimally aligned virtual trajectory
- Haversine alignment procedure
- Align desired subset of atoms
- Apply alignment transformation to frame (or subset)