On Developing Reusable Software Components for the Advanced Cyberinfrastructure

October, 2019 | Ritu Arora
XSEDE ECSS Symposium | Email: rauta@tacc.utexas.edu
What is software reusability, and why should one care about it?

• Reusability refers to creating new software products using previously existing software components or other software artifacts

• It saves cost and time in re-implementing or re-inventing something that already exists

• Examples of software reuse: functions and subroutines, Standard Template Library (STL) in C++
What does software interoperability mean?

- Ability of two or more software components or systems to interact with each other and exchange usable information.

- It is a key property that is evaluated during the process of integrating the software components from different developers or groups.

- Example: off-the-shelf chatting software integrated into online air ticket reservation systems.

Photo source: https://adarvetranslations.com/2014/06/05/interoperability-cat-tools/
Why should scientific software developers care about reusability and interoperability? (1)

- Reusability and interoperability can enhance the quality of the software

- Students eventually graduate, and developers get promoted to become managers
  - What happens to the home-grown software that they leave behind, which may or may not be well documented or of high-quality?

Why should scientific software developers care about reusability and interoperability? (2)

- The software products that are reusable and interoperable tend to be well-documented, and vice-versa
  - Hence, on-boarding the new team members in the absence of the original developers becomes easy

- Reusability and interoperability may be important for the funding agencies that are evaluating your project proposal
Why should scientific software developers care about reusability and interoperability? (3)

• Scientific software and the methods associated with them may end-up being used in unforeseen circumstances
What is IPT?

Interactive Parallelization Tool (IPT) is a semi-automatic tool that converts a C/C++ serial program into an efficient parallel program by parsing the specification by the users. It is currently being used for teaching parallel programming and supports MPI, OpenMP, and CUDA programming models.

You can use IPT from the convenience of your web-browser. It is simple. Start by creating an account on this website or use your existing TACC portal account information.

If you login with your TACC portal account credentials, you will be also be able to compile and run the program generated by IPT on two XSEDE systems: Stampede2 and Comet.
VOLUNTEER COMPUTING

Contribute your extra computing power to open science. By running BOINC while your system is idle you will be powering discoveries that change the world.

JOIN BOINC@TACC

What is BOINC@TACC?
It is a project based on the Volunteer Computing (VC) model. It helps TACC/XSEDE researchers in running applications from a wide range of scientific domains (such as, aerospace engineering, computational biology, and earthquake engineering) on the laptops, desktops, tablets, or cloud-based Virtual Machines (VMs) owned by volunteers.

How can volunteers help?
Volunteers can help scientific research by signing up for the BOINC@TACC project and allowing it to run applications on their computers and VMs. They may also need to download the required software from this website. To volunteer, join BOINC@TACC after reading the project policies.

How do researchers benefit?
This project helps in supplementing the computing cycles available through their TACC/XSEDE allocations. With BOINC@TACC, researchers can run small high-throughput computing jobs that involve small amounts of data transfer and processing without spending their active allocations. For details on using the BOINC@TACC infrastructure, click here.

How is BOINC@TACC implemented?
It leverages the middleware and extends it to support the job submissions from supercomputers, uses the VMs in the cloud, and automatically creates Docker images of the source code written in selected languages. The project source code is available on Github.
Greyfish

To address the file storage and quota needs of BOINC@TACC and GIB (IPT web portal) projects, we developed a portable, cloud-based filesystem with the interfaces to upload and download files, admin features such as purge and scanning directories and files, create user accounts, bulk upload, and bulk download.

Source Code: https://github.com/ritua2/greyfish
Job Submission

Location of docker image *
- List of Docker images maintained by BOINC@TACC
- Docker hub
- Automated docker build

Operating System *
- List of operating system supported by TACC-2-BOINC
- Python3 (We are not currently supporting Python 2)
- R
- C
- C++ (Please refer to this cget package manager for the libraries)
- Fortran
- Bash

Programming languages and libraries *
- Setup file needed
- No setup file needed

Browse

Setup File *

Browse

Browse

No file chosen

The Job Topic and/or Sub Topics
Format: TOPIC(SUBTOPICS), e.g., BIOLOGY(GENETICS, ANATOMY);COMPUTER SCIENCE()

Commands *
e.g., gcc -o hello.exe hello.c (hit enter at each of the end the command line including the last command line)

Output Files *
- Get all output files
- Get only some output files

Input files *
- Tar Upload
- Zip Upload
- No Input Files

Browse

Browse

No file chosen

Submit
IPT exhibits the property of software reuse too

- IPT allows users to compose their own design patterns for parallel communication and computation using existing design templates.

- Each design template is written to do just one type of activity.

- These templates can be composed to form higher level patterns and can be reused multiple times while parallelizing applications from different domains.
Why was it important to incorporate the property of reusability in IPT?

- Community-building is a requirement for the projects funded through the NSF SI2 program
- One-size-fits all approach does not work for community-building
  - While IPT and its deployment in the cloud maybe useful for beginners in parallel programming, the intermediate or advanced-level users may be interested in extracting the design templates that capture the parallel communication and computation patterns

Photo source: https://www.sunset.com/garden/backyard-projects/propagate-succulents
Lessons Learnt (1)

• It is hard to write reusable code that someone else can use
  – Designing the interfaces and building a community that has sustained interest in reusing the software are not easy to do, plus, these are time-consuming activities

• It is hard to reuse someone else's code
  – While reuse works fine with operating systems, libraries, and middleware, people often struggle to reuse software developed by others
  – Some people (myself) have a mental barrier towards reuse and may feel that reuse would make them dependent on somebody else

References:  https://www.dre.vanderbilt.edu/~schmidt/reuse-lessons.html
Lessons Learnt (2)

• Reuse can create a dependency on something you don't own, know, or maintain
  – Such a dependency may mean that you may no longer be in control of the entire software development lifecycle
  – The feature or change that you may want to implement may not be compatible with the interfaces of other components with which your software could be interoperating with

• Workaround?
  – Allow people to use your software at their own risk
  – Select a license that allows others to fork from your project and make changes as needed to the code base
Lessons Learnt (3)

• A thorough interoperability analysis is critical before integrating software components and services, especially those involving black-box integration

• Cost-benefit analysis is critical too
  – There can be personnel changes - the other software that your software should interoperate with, may not be maintained any further
  – It is possible that you may not find someone who understands the tools and technologies involved in maintaining the other software or integrating it with your software
When to focus on reuse and interoperability and when not to worry about it?

- For small-scale projects (viz., IPT), it may be economical to build from scratch instead of relying on integration with other products.

- When one needs the fine-grained ability/control to tweak the details that may affect the performance and desired output of a software, it may be better to build from scratch than reuse.

- For large-scale projects having 1) a large community, 2) related objectives, and 3) sufficient resources, it may be better to form a technology integration and assessment team, and develop reusable software components that can leverage from each other's investments.
Conclusion

• There is often a direct or indirect cost involved in making software reusable and interoperable, and this cost may vary from project to project
  – Developing well-document and reusable software components can have a significant impact on the software quality
• Before reusing or interoperating with other software, consider the following:
  – installation overheads
  – dependency on the project personnel/changes
  – software maintainability
  – change management
  – existence of shared documents about interoperable software units with high-level architecture to help in the analysis
Thanks for your kind attention!

The IPT project is sponsored by NSF SI2:SSE Award # 1642396.

The BOINC@TACC project is sponsored by NSF SI2 Award #1664022.

Email: rauta@tacc.utexas.edu