Extended Abstract: Building applications from definition files with GenApp

Emre Brookes* and Alexey Savelyev
Department of Biochemistry, The University of Texas Health Science Center
at San Antonio, 7703 Floyd Curl Drive,
San Antonio, TX 78229-3900
**Corresponding author:* email: emre@biochem.uthscsa.edu

Abstract: *GenApp is a tool for rapid deployment of applications. GenApp builds fully functioning applications from collections of definition files and libraries of code fragments. Each application requires two global definition files, which describe the application and the menu structure. There are any number of module definition files. Each module definition file fully describes the interface to the underlying executable. Each target language, equivalently the language of the produced application, also has a definition file which describes how the target language code fragments are assembled and transformed into the produced application. This structure enables target language agnostic development, helping insure preservation of underlying executables in an ever changing software environment landscape. Interesting questions arise about how best interactivity can be defined in a module definition file. To address interactivity, we have not only included advanced field types such as plots and atomic structures, but have also added calculated relationships between fields and introduced the concept of "repeaters", which can provide dynamic field content.*

1. Introduction

The GenApp [1,2,3,4] framework is designed to simplify creation and deployment of local GUI and web based applications over a collection of modules. GenApp was originally conceived as a tool to build applications for the jointly NSF and UK's EPSRC funded CCP-SAS project [5] wrapping a variety of underlying codes [6,7,8]. GenApp is successfully used in these and other cases. This presentation will describe details of the structure of GenApp, how it builds applications and how GenApp facilitates definition file based interactivity.

Presented at Gateways 2016, San Diego Supercomputer Center, La Jolla, CA, November 2-3, 2016.
https://gateways2016.figshare.com/

2. GenApp Structure

2.1 Roles

GenApp defines four primary roles. These are, in order of descending C.S. expertise, the framework developer, the target language developer, the application developer and the module developer. These roles parallel the organization of GenApp as shown in Fig. 1. The framework developer develops and maintains the generator tool. The target language developer is responsible for the contents of one or more target languages and implements and maintains them by building up code fragments and defining their assembly. The application developer organizes modules in a menu definition file and runs the GenApp generator to create working applications. Finally, the module developer wraps executable modules by writing a module definition file and ensuring the wrapped executable accepts defined input and produces defined output.

This structure segregates the application and module developers, which require minimal CS expertise from the framework and target language developers, which require advanced CS expertise. This enables a researcher, acting as an application and module developer, with an executable that they wish to expose to rapidly deploy advanced user interfaces. If a new target language, variant or feature is developed by a target language developer, it becomes available to all application and module developers and they can deploy the new target.

2.2 Modules

A module is some defined executable within GenApp. The module definition file (Fig. 1) contains all information about the module. This, of course, includes all input and output fields. Each field is uniquely defined with an *id*. In addition, a

primary attribute for each field is the *type*, e.g. “integer”, “text”, “plot”, “atomicstructure”, etc.

2.3 Repeaters

Early on, module developer requirements presented the necessity to define input fields which appear to the user based upon the values of other input fields. The GenApp answer to this was the *repeater* structure. For example, a “checkbox” field could have the attribute of *repeater*, subsequently, other fields could reference a *repeat* on the *repeater*'s *id*. This enables a field to be displayed dynamically to the user based upon the setting of the “checkbox”. Current *repeater types* include “checkbox”, “integer” and “listbox”. The “checkbox” *repeater* has been described and a reverse logic “checkbox” *repeater* is also available. The “integer” *repeater* will create some number of instances of the dependent *repeat* fields. The “listbox” *repeater* displays dependent fields based upon the “listbox” choice. Repeaters can be arbitrarily nested.

2.4 Calculated Fields

Another module developer requirement was the calculated field. In this case, an input field is computed based upon values in other input fields based upon a mathematical equation. This puts the computation work within the user interface which dynamically updates values. Calculated fields can be based upon values of other calculated fields and calculations can occur under *repeaters*.

3. Conclusion

GenApp is a tool for rapid deployment of applications. Wrapping an executable as a module can help preserve the module for future target languages. Each executable is defined in a module file which can describe dynamic content handled fully with the user interface.

4. Acknowledgments

This work is supported by the NSF SSI grant CHE-1265817 to E. Brookes.

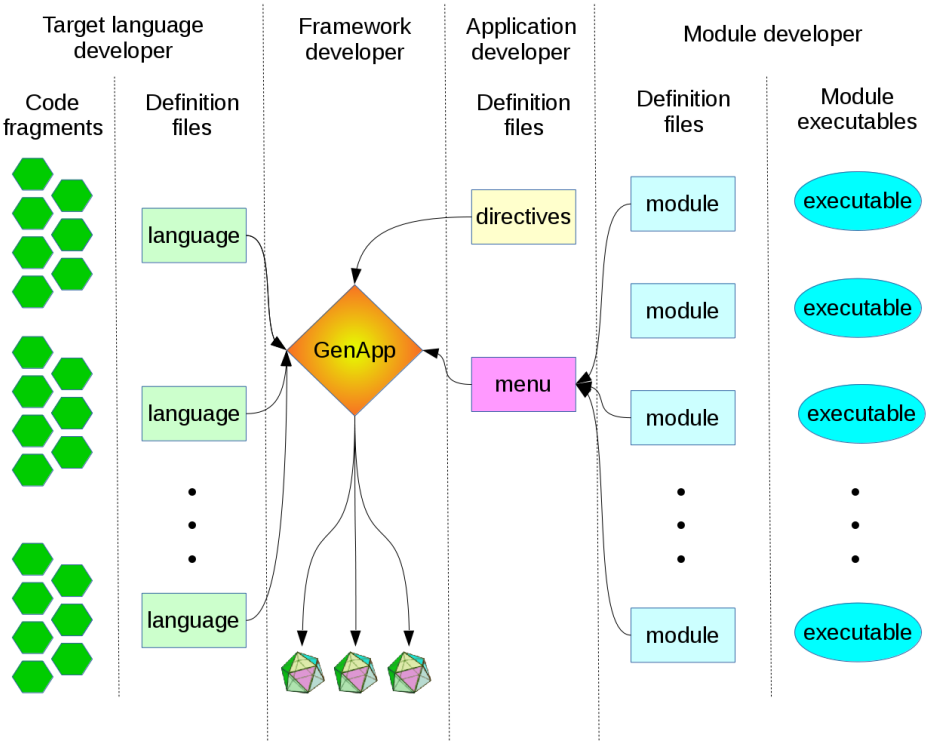

Fig. 1. An overview of the GenApp framework. Four roles are defined at the top as areas of responsibility. The application and module developers are simple roles and are appropriate for researchers without CS expertise. The target language and framework developers require a much higher level of CS expertise. This separation allows researchers to rapidly deploy applications in a variety of target languages and take advantage of new developments within the framework without effecting their underlying executable.

5. References

[1] Brookes, E.H. 2014. *An Open Extensible Multi-Target Application Generation Tool for Simple Rapid Deployment of Multi-Scale Scientific Codes.* XSEDE '14. ACM, doi: 10.1145/2616498.2616560

[2] Brookes, E.H., Anjum, N., Curtis, J.E., Marru, S., Singh, R., and Pierce, M. (2015), *The GenApp framework integrated with Airavata for managed compute resource submissions.* Concurrency Computat.: Pract. Exper., 27(16):4292-4303, doi: 10.1002/cpe.3519.

[3] GenApp. http://genapp.rocks

[4] Brookes, E.H., Kapoor, A., Patra, P., Marru, S., Singh, R., Pierce, M. (2015) *GSoC 2015 student contributions to GenApp and Airavata,* Concurrency Computat.: Pract. Exper., 28(7):1960-1970, doi:10.1002/cpe.3689

[5] Perkins, S., Butler, P., *CCP-SAS – Collaborative Computational Project for advanced analyzes of structural data in chemical biology and soft condensed matter.* http://ccpsas.org

[6] Curtis, J. E, Raghunandan, S., Nanda, H., and S. Krueger. (2012) *SASSIE: A program to study intrinsically disordered biological molecules and macromolecular ensembles using experimental restraints.* Comp. Phys. Comm.183:382-389. http://www.smallangles.net/sassie

[7] Brookes, E., Demeler., B, and Rocco, M. (2010). *The implementation of SOMO (SOlution MOdeller) in the UltraScan analytical ultracentrifugation data analysis suite: enhanced capabilities allow the reliable hydrodynamic modeling of virtually any kind of biomacromolecule*. Eur. Biophys. J, 2010 doi:10.1007/s00249-009-0418-0

[8] Wright, D. and Perkins, S., SCT software, http://dww100.github.io/sct/