Component-based Workflow Framework for Data Analysis

Marko Laakso*, Kristian Ovaska*, Sampsa Hautaniemi
Computational Systems Biology Laboratory,
Institute of Biomedicine and Genome-Scale Biology Program,
Biomedicum Helsinki, University of Helsinki
**equal contribution*

April 9, 2009

Abstract

We provide the framework Anduril (ANalysis of Data Using Rapid Integration of aLgorithms) for systematic methodology development, adoption of methods developed by third parties and rapid analysis of data. Anduril architecture is built on generic, reusable components, such as algorithms and connections to remote databases. The components comprise a network that allows rapid and reproducible analysis, straightforward inclusion of novel methods and comprehensive summary report including the results and parameters for the components. In our case study we demonstrate the utility of Anduril in testicular cancer study. Anduril is platform-independent and is available under the GNU General Public License from `http://csbi.ltdk.helsinki.fi/anduril/`.

1 Introduction

Computational analysis and integration of data from heterogeneous sources are essential features of systems biology. In particular the role of multivariate methodologies is prominent when analyzing high-throughput data sets typical for genome wide studies. Applications of such analysis methods typically produce long lists of candidate genes or proteins that are often challenging to translate into novel knowledge or testable hypotheses. Thus, integration of data from heterogeneous sources, such as protein-protein interaction or pathway databases, are used to gain putative functional consequences of the candidate gene lists. Indeed, bioinformatics community has developed a vast array of methodologies to analyze and integrate data, and demonstrated their utility in several diseases and model organisms. The availability of such methodologies creates a need for a framework that allows systematic and flexible integration of data and analysis tools.

Here, we describe Anduril (ANalysis of Data Using Rapid Integration of aLgorithms), an open source framework for advanced bioinformatics analysis. Anduril is able to control the whole workflow from data importing to automatically generated summary report including parameters for the methods executed in the workflow, figures, tables, and annotation information. A workflow in Anduril is based on execution of components that are entities responsible for specific data processing tasks. The component based architecture allows straightforward integration of novel algorithms and services.

There exist few workflow-based systems for bioinformatics analysis, most notably GenePattern [5] and Taverna [4]. In comparison to these methods, Anduril offers several unique features. For example, automatic execution of only those components whose configuration has been changed since the previous run may result significant time saving in large-scale projects. Also, automatic generation of the summary report for the whole workflow facilitates interpretation of the results. Anduril has been designed primary for local installations where large data sets can be moved between the components and the development and the testing of new methods plays a significant role.

The design principles for Anduril stem from needs to manage large-scale, bioinformatics heavy projects. In such projects the analysis is performed by a team of bioinformaticians rather than an individual bioinformatician. The analysis cycle includes executing new analyses with different parameter settings or novel algorithms based on discussion with experimental bioscientists. This kind of interactive analysis forms the core of systems biology and requires a flexible bioinformatics framework that allows concurrent programming of the methods with, for instance, the agile programming development principles.

Anduril is designed for bioinformaticians and computational biologists familiar with command line interfaces and preferably having some programming experience. The end product of an analysis is a summary report that can be used as the basis of the results evaluation. The summary report can be used as an article supplement to provide details of the analysis steps so that the whole analysis is truly reproducible.

2 Methods

Anduril workflows consists of steps of analysis that are following each other. Each step is carried out by a component capable of carrying it out. A component is an executable that reads files as input and writes files as output. A component can have varying number of mandatory or optional input and output *ports*. Each port represents a file or a directory. In addition to file inputs, components have simple parameters such as Booleans, integers, and strings. The ports and *simple parameters* have types associated with them. The type system enables static error checking and component interface documentation. The port types are extensible, while simple parameter types are fixed. The type system supports subtyping, so that a component that supports type A will also accept type B if B is a subtype of A. The interfaces of components are documented in a uniform manner, so components can be used as black-boxes when necessary.

Components can be reused between analysis projects but it is also possible to write project-specific components that are used only in one analysis. Most components are provided by application-area specific frameworks and the backbone framework provides a few core components.

Components can be written in any language since the only requirement is the ability to read and write files. Language independence enables to leverage existing code from various libraries such as the Bioconductor library for R [2] or other community-effort services. In order to facilitate component writing, we provide language specific mini-frameworks for the languages such as Java, Matlab, Python and R. Mini-frameworks provide services that most components need, such as an interface to the workflow engine and facilities for reading and writing files associated to ports.

A component is used through its external interface, which consists of input and output ports and simple parameters. An important part of the interface is documentation. All aspects of the interface can be documented. The interface is specified with an extensible markup language (XML) file called a *descriptor file*. The framework creates HTML manual pages for all components based on descriptor files. This provides uniform documentation for components written in different languages.

Type parameters can be used to increase the flexibility of components. A *type parameter* is a placeholder for a type that is fixed when the component is used in a workflow. A component that has type parameters is called a generic component. A type parameter can be used for input and output ports in place of a concrete type. The mechanism is similar to generics in Java. An example of a generic component is one that copies a file from one location to another. It does not care about the type of the file and therefore the type is parametrized. The actual types for type parameters can usually be assigned by a type inference algorithm, so much of the generics machinery works transparently to the user.

Concrete applications are created by wiring selected components into a directed network where the output port of a component is connected to the input port of another. Since the ports are typed, the framework is able to statically detect type mismatches in port connections. Static error checking is advantageous in large-scale projects where execution of an analysis pipeline may take days.

Composite components allow encapsulation of several components into one and modularizing a large network into subnetworks. A *composite component* is a network of components that functions as a single component. Each component implements a part of the composite component.

A composite component can be instantiated several times in a network. Like regular components, composite components may have any number of input and output ports and simple parameters. In addition to help in managing large workflows, composite components allow to reuse network topologies between analysis projects, much like function libraries in traditional programming languages.

Conditional branch components are used to dynamically select alternative execution paths in the network. They correspond to case statements in programming languages. A conditional branch component is associated with two or more optional branches and it can enable each branch independently. At least one branch must be enabled. Each conditional branch has a unique associated join component that joins the branches.

Network topology is configured using a high-level domain specific language (DSL) that syntactically resembles programming languages such as Java but is much simpler. The advantage of a DSL is that it is possible to use the framework without writing code in an actual programming language. The language is strongly and statically typed [6], which means that the types of all names are known before the network is executed and type errors are caught. Types are not explicitly declared but are inferred from the context. Accepted values of parameters are determined by their types (*boolean*, *int*, *float*, *string*) but cannot be further restricted to certain ranges. Syntax of the configuration file has been designed for humans working with text editors and integrated development environments. Each component instance is represented by a simple call that defined its inputs and parameters and an assignment to the instance name.

The network is executed by launching components in any order permitted by connection dependencies. Components that do not depend on each other are automatically executed in parallel. This allows to take an advantage of multi-core CPUs without requiring the user to take care of details such as process creation and synchronization. The maximum number of parallel threads can be configured. The current version of Anduril does not support task queues commonly used in computer clusters but this would be a useful target for development in future.

All intermediate and final results of components are stored on disk. Also, configuration settings of all components are stored. When a network is executed again, the workflow engine can infer components whose configuration has changed since the previous run. A component is considered as changed if its simple parameters, input connections, version number or timestamps of input files have changed. Only changed components and those that depend on changed components are re-executed. This partial re-execution of the network supports the iterative nature of scientific research. The researcher can tweak parameters of components to see their effect on the result and only affected components are executed. Partial re-execution is transparent to the user. As a consequence of this feature, each component instance must be executed at most once and the network must be acyclic. Component instances that are intended for repeated invokes, such as some those generating timestamps or responsible of fetching data from volatile sources, can be tagged for automatic refresh on each execution. It is also possible to freeze certain instances so that they do not react to changes in their inputs. Although this may risk the validity of the results it helps in restricting the effects of alterations of certain 'construction sites' from propagating unnecessarily.

Anduril provides support for component-level and network-level black box testing that aims at ensuring that components and workflows are working correctly. Test cases also provide a convenient method to invoke components that are under development, without the need to place the components into actual analysis networks. Test-driven development, where component test cases are written before code, is an approach that has worked well in various large-scale software projects [1].

3 Results

To demonstrate the usefulness and features of the framework we have re-analyzed microarray expression data from a testicular germ cell tumor study [7]. Testicular germ cell tumor (TGCT) is the most common type of germ cell tumors [7]. In testicular tumorigenesis normal germ cells first develop to premalignant and noninvasive intratubular germ cell neoplasia (IGCN). Based on histology, TGCTs are categorized as seminoma, which resemble IGCN, and non-seminoma that are a heterogeneous group of tumors [7]. The embryonal carcinoma cells can further differentiate to choriocarcinoma, teratoma and yolk sac tumor. The main objective of the study by Skotheim et al. was to compare expression profiles between undifferentiated embryonal carcinoma and differentiated samples (normal testicular parenchyma and the differentiated subtypes of nonseminomas, which include teratoma, choriocarcinoma, and yolk sac tumor). Here, our goal is the same but using alternative and additional methods. Quality control of the input data and the result annotations are considerably improved in comparison to the original analysis.

Configuration and the results are reported automatically after each execution. Full Anduril generated summary report can be found from Anduril home page. Also the network and components used in this case study are freely available. In total, the network is composed of more than 100 component instances, some of which are encapsulated in composite components. Input data consists of 17 samples analyzed with two-color Agilent Human 1A microarrays that measure 19061 features. The reference in each experiment was a pool of 10 human cell lines. Network execution takes approximately 15 minutes on a contemporary server when the network is executed for the first time. On the subsequent runs, execution took between 15 seconds and 15 minutes depending on the changes made to the network configurations.

4 Discussion and conclusions

Anduril is a unique workflow-based framework for professional data analysis. The architecture and features are designed to support systematic methodology development, adoption of methods developed by third parties and rapid analysis of data. Thus, Anduril also provides means to describe course of the analysis with sufficient details rendering the analysis truly reproducible. This feature is crucially important as a major reason for failure to validate some cancer diagnostic studies is attributed to insufficient description of the analysis [3].

In our case study we reanalyzed the data originating from Skotheim et al. [7] to demonstrate the utility of Anduril and its microarray analysis framework. The running of the whole analysis including pathway and protein-protein interaction analyses took approximately 15 minutes. To a large degree, our results agreed well with Skotheim et al. despite the use of different preprocessing and methods. For example, the genes *GAL* (Galanin Precursor) and *POU5F1* (POU domain, class 5, transcription factor 1) were also identified by us and these genes were validated as diagnostic markers for undifferentiated tumor cells using tissue microarray with 510 clinical testicular samples. We have also demonstrated that Anduril facilitates testing different parameter values, the impact of exclusion of a subset of samples and integration of novel algorithms. Though detailed discussion of the results is beyond the scope of this study, the ability of Anduril to rapidly analyze high-throughput data should be clear.

Currently, we provide over 100 components mostly for high-throughput data analysis. The component repository is freely available and updated frequently. Further, inclusion of new components to Anduril is straightforward and the capability to harness novel algorithms allows analysts to focus on improving existing algorithms or developing novel ones rather than reinventing the wheel. We are currently developing an Eclipse plug-in for Anduril. This plug-in will provide syntax highlighting, syntax validation and the selection of supported components and connections. The software is freely available on-line and includes a comprehensive user guide and documentation.

References

[1] K. Beck. *Test-driven Development: By Example.* Addison-Wesley Professional, 2003.

[2] Robert C Gentleman, Vincent J. Carey, Douglas M. Bates, et al. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004.

[3] J.P.A. Ioannidis, D.B. Allison, C.A. Ball, I. Coulibaly, X. Cui, A.C. Culhane, M. Falchi, C. Furlanello, L. Game, G. Jurman, et al. Repeatability of published microarray gene expression analyses. *Nature Genetics*, 41(2):149–155, 2008.

[4] T. Oinn, M. Greenwood, M. Addis, M.N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, et al. Taverna: Lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, 2006.

[5] M. Reich, T. Liefeld, J. Gould, J. Lerner, P. Tamayo, and J.P. Mesirov. GenePattern 2.0. *Nat Genet*, 38(5):500–501, 2006.

[6] M.L. Scott. *Programming Language Pragmatics.* Morgan Kaufmann, 1999.

[7] R.I. Skotheim, G.E. Lind, O. Monni, J.M. Nesland, V.M. Abeler, S.D. Fossa, N. Duale, G. Brunborg, O. Kallioniemi, P.W. Andrews, and RA Lothe. Differentiation of human embryonal carcinomas in vitro and in vivo reveals expression profiles relevant to normal development. *Cancer Research*, 65(13):5588–5598, 2005.