SOFTWARE

Building applications for interactive data exploration in systems biology

Bjørn Fjukstad[1], Vanessa Dumeaux[2], Karina Standahl Olsen[3], Eiliv Lund[3] and Lars Ailo Bongo[1*]

Abstract

Background: In scientific fields such as systems biology there is a need for interactive data exploration tools to enable new insights in the fast growing datasets. These tools combine advanced statistical analyses, known biology from up-to-date databases, and visualizations. The problem solved by each tool differs, but typically their underlying components are similiar. Application developers can therefore compose applications of reusable services rather than implementing a single monolithic application.

Results: We have designed an approach for developing data exploration applications in systems biology that builds on the microservice architecture. We use this approach to build applications that integrate advanced statistical software and up-to-date information from biological databases. We demonstrate its viability through a web application for exploring and comparing transcriptional profiles from blood and tumor samples. In addition we have used it to build two other web-applications and several command-line tools. With a microservices approach we can re-use and share key components between application reducing development, deployment and maintenance time.

Conclusions: Our approach and reference implementation Kvik is open-sourced at github.com/fjukstad/kvik and the web application for exploring transcriptional profiles, MIxT, is availible at github.com/fjukstad/mixt.

Keywords:

Background

In the past decade the generation of biological datasets has been unprecedented, and the famous "$1000k genome, $1M analysis"[] has become more apparent.

To analyze the growing biological data sets, there is now a number of analysis tools in various programming languages.[] In R, there are popular package repositories such as CRAN cran.r-project.org or Bioconductor bioconductor.org where developers can share software packages and keep them up-to-date. These repositories contain software packages for exploring high-throughput genomic data in one environment. This includes both pre-processing, e.g. cleaning, removing outliers, and analysing it with known statistical methods. In other languages such as Python or Go developers can choose bioinformatics libraries such as BioPython[] and biogo[] respectively. To use these packages to their full potential researchers require a high level of coding skill, and However all of these software packages require users to have a high level of coding skills.

A key part of analysing biological data is to visualize it. Researchers often start data analysis by using simple visualization techniques to get a quick overview of the data. Continuing in the data analysis pipeline researchers can use more advanced visual techniques to explore the data either using software packages or complete data visualization applications. Traditionally data visualization applications have been built as desktop applications that require installation and setup by the user, but now the move is towards software that run in the web browser without any user setup.

To visualize and share results from statistical analyses, the results are often exported to a data format such as comma-separated values (CSV) and then visualized using an external tool. This decouples data presentation and analysis. Through initiatives such as ApacheR and OpenCPU there has been a move towards embedding scientific computation in data visualization application. This removes the decoupling of statistical frameworks and interaction with the analysis results.

*Correspondence: larsab@cs.uit.no
[1]Department of Computer Science, UiT – The Arctic University of Tromsø, 9037 Tromsø, NO
Full list of author information is available at the end of the article

Interpreting the analysis results require integration of known biology, either from biological databases such as MSigDB[] or KEGG[], or through scientific publications from reference databases such as PubMed[]. Performing manual lookup into these databases is often tedious and error-prone, making it necessary to automate the task. Now as more databases provide REST APIs it is possible to provide software packages for automatic retrival of database information. In addition, since databases are continuously being updated, using a REST API to retrieve database information will ensure that the data is always kept up to date.

Microservice architectures structures applications into small reusable, loosely coupled parts. These communicate via lightweight programming language-agnostic protocols such as HTTP, making it possible to write single applications in multiple different programming languages. This makes it possible to use the most suitable programming language for each specific part. E.g. to use R and Bioconductor to analyse biological data, or C++ and OpenCV for high-performance computer vision tasks, or HTML, CSS, and JavaScript to build portable user-interfaces. To build a microservice application, developers bundle each service in a container that are deployed. Containers are built from configuration files which describe the operating system, software packages and versions of these. This makes reproducing an application a trivial task. The most popular implementation of a software container is Docker[1], but others such as Rkt[2] exist.

Requirements

From our experience building data exploration applications we have identified a set of reusable services that an application developers can use to build a wide range of applications. The key services of a biological data exploration application are i) a service for executing for statistical analyses in languages such as R, and ii) a query service for retrieving meta-data on genes or other biological entities. On top of these services is possible to build any number

To build these services we need a framework that fulfills the following requirements:

1 It provides language-independent approach for integrating, or embedding, statistical software, such as R, directly in interactive data exploration applications.
2 It profices an interface to online databases to provide meta-data to biological entities.
3 A framework with components that are easy to develop, maintain and deploy.

[1]

[2]

Contributions

Our contributions are:

1 An approach for developing data exploration applications for systems biology that combine statistical analyses with online databases.
2 A demonstration of its viability through N different applications.
3 Performance evaluation of its central data analysis component.

Related Work

In this section we aim to cover the existing approaches for building interactive data exploration applications in systems biology.

OpenCPU

OpenCPU is a system for embedded scientific computing and reproducible research.[**?**] It provides an HTTP API to the R programming language to provide an interface with statistical methods. It enables users to make function calls to any R package and retrieve the results in a wide variety of formats such as json or pdf. Users can chose to host their own R server or use public servers, and OpenCPU works in a single-user setting within an R session, or a multi-user setting facilitating multiple parallel requests. OpenCPU provides a Javascript library for interfacing with R, as well as Docker containers for easy installation. OpenCPU has been used to build multiple applications.[3]. In Kvik we provide a package to interface with OpenCPU servers from the Go programming language since it follows the design pattern we have chosen to interface with data and analyses.

Renjin

Renjin is a JVM-based interpreter for the R programming language.[**?**] It targets developers who want to integrate the R interpreter in web applications. Since it is built on top of the JVM it allows developers to write data exploration applications that interact directly with R code. Although Renjin supports a large number of CRAN packages it cannot access any R package (e.g. from BioConductor) without modification. This makes the programming effort to use Renjin as an interface with R higher.

Shiny

Shiny is a web application framework for R.[**?**] It allows developers to build web applications in R without having to have any knowledge about HTML, CSS or Javascript. It provides a widget library to provide more advanced Javascript visualizations such as Leaflet for

[3] opencpu.org/apps.html

maps or threejs for WebGL-accellerated graphics. Developers can choose to host their own web server with the user-built Shiny Apps, or host them on public servers. Shiny forces users to implement data exploration applications in R, limiting the functionality to the widgets and libraries in Shiny.

Biogo
Biogo is a bioinformatics library in Go. It provides functionality to analyse genomic and metagenomc datasets in the go programming language.[**?**] Using the go programming language the developers are able to provide high-performance parallel processing in a clean and simple programming language.

Cytoscape
Cytoscape is an open source software platform for visualizing complex networks and integrating these with any type of attribute data[**?**]. It allows for analysis and visualization in the same platform. Users can add additional features, such as databases connections or new layouts, through Apps. To bring the visualization and analysis capabilities to the web the creators of Cytoscape have developed Cytoscape.js[4], a Javascript library to create interactive graph visualizations.

cyREST.

Methods

In this section we first motivate our microservice approach by describing how we developed a web application for exploring and comparing transcriptional profiles from blood and tumor samples, called MIxT Blood-Tumor. We describe the process from initial data analysis to the final application, highlighting the importance of language-agnostic services to facilitate the use of different tools in different parts of the application. This is especially important in interdisciplinary teams where researchers use a wide range of tools. After the description of our approach we generalize the ideas to a set of principles and services that can be reused and shared between projects. We believe many system biology data exploration applications are developed similarly and that they can therefore also benefit from the microservice approach.

Analyzing biological datasets

We analyzed profiled RNA in blood and matched tumor from 173 patients in the Norwegian Women and Cancer (NOWAC) study. Each profile measures the expression of 16 782 unique genes.

We used R and various packages from Bioconductor and CRAN to pre-process and analyze the gene expression dataset. From the initial analyses we built an R package that makes it possible to share results between researchers within, and outside our group. To build an application on top of the analyses there are two possibilities: i) develop a script that generates a static output file (e.g. generate a html report or CSV files that contain the result data) and build an application that uses this static data; or ii) use a framework such as Shiny or OpenCPU to create a dynamic application that interface directly with the R code. To update an application that builds on top of a static output file requires manual execution of the output script, while using Shiny or OpenCPU this task.

We typically start off with a messy dataset that needs to go through several stages of clean-up and preprocessing before we can analyze it. After the preprocessing we typically develop some simple visualizations that help discover simple patterns in the data. After this quick dirty data exploration we start to apply more advanced statistical methods to look for more intricate patterns in the data. After this analysis we often end up with genes or lists of genes of interest that we can use to guide database lookup.

In terms of data analysis code, the preprocessing steps typically consist of one or more R scripts that we knit [**?**] into PDF reports that we can revisit later. From these scripts we end up with analysis-ready datasets that can be shared within the group. The remaining downstream analysis often starts out in scripts, that are built into R packages with analysis code that can be shared between researchers.

0.1 Query reference databases

0.2 Build an application

0.3 Kvik micoservices

Based on the development of MiXT and other data exploration tools, we have generalized our experience into the following design principle guidelines and microservices provided by the Kvik framework:

Principle 1: language-agnostic (samme har de funnet ut i blant annet Facebook for Thrift) Principle 2:

Microservice 1: Databases... Microservice 2: Statistical analyses...

With this process in mind, we designed the interface to the R programming language in Kvik. We want to make it possible to call any function from an R package and return its results either as plain text, such as comma-separated tables, or binaries such as images. Enforcing that R code is built into R packages ensures that the analysis code can be used by power users through an ordinary R session or in the data exploration application itself.

[4] js.cytoscapejs.org

Databases

Similar to how our analysis process shaped the R interface, it also defined how we want to build interfaces to online databases.

In its initial state we wanted an interface to interactively query databases such as KEGG or MSigDB for up-to-date information about genes, gene sets or biological pathways. This interface should be available within the data exploration applications to provide valuable metadata, such as gene summaries, for the researchers exploring results.

Building applications

With Kvik there are multiple ways developers can build data exploration applications. Either bundle analysis and database lookup on a single computer, or separate computational tasks to more powerful compute clusters to improve performance. In this paper we discuss how to develop applications that follow a microservices architecture where data analysis and storage is simply a service.

In Kvik we use R packages as the fundamental building block for data exploration applications. They provide an interface to data and analyses, and especially in the field of systems biology, the R programming language provide the largest collection of data analysis packages. We discovered that the most sensible way to build applications on top of our existing code base was to build a system that could interface with our analysis code directly. In Kvik we built an HTTP interface on top of R that allows users to call functions and get results using any programming language with an HTTP library. This allows developers to build data exploration applications in the programming language that is most suitable, or has the best support, for presenting that specific data type.

Statistical analyses

Describe how we've designed the interface with R: Build an R-package and call functions from it, we provide four different output formats to the user (json, csv, pdf, png), as well as four different http endpoints (call, get and rpc).

The R interface in Kvik follows many of the design patterns in OpenCPU. Both systems interface with R packages using a hybrid state pattern over HTTP. Both systems provide the same interface to execute analyses and retrieve results. While OpenCPU is implemented on top of R and Apache, Kvik is implemented from the ground up in Go. Because of the similarities in the interface to R in Kvik we provide packages for interfacing with our own R server or OpenCPU R servers through the *gopencpu* package.

The R server in Kvik builds on the standard *http* library in Go. On start it launches a user-defined number of R sessions that execute analyses on demand. This allows for parallel execution of analyses. We provide a simple FIFO queue for queuing of requests. The R server also provides the opportunity for users to cache analysis results to speed up subsequent calls.

The Kvik R server is suitable for applications where the analysis should be run on a different server than the web-server hosting the application. If users want to bundle both the application and R server on the same machine, the *r* package in Kvik provides this functionality. Although this is possible, we believe that following a modular approach separating analysis and application user-interface makes a cleaner and easier to maintain application.

Databases

Describe the interface to the databases and what we use it for. Could be interesting to talk about provenance/caching.

Implementation

Applications

Results and Discussion

Describe the MIxT application. Also talk about how our R interface scales and what makes it better than opencpu/renji.

Use Case

We show the viability and need for Kvik by describing the MIxT application for exploring and comparing transcriptional profiles from blood and tumor samples. We describe its functionality, implementation and performance requirements. Then we describe how MIxT is designed to separate concerns and allow for a layered implementation. We use this to motivate the need and opportunities to abstract away common functionality of these type of applications.

Matched Interaction Across Tissues (MIxT)

We have built a system to identify genes and pathways in the primary tumor that are tightly linked to genes and pathways in the patient systemic response[**?**]. MIxT blood-tumor is an open-source web application for exploring the molecular processes expressed in each tissue.

For the web application we defined six analysis tasks:

Explore co-expression relationships between genes. Create an interactive network visualization that visualizes each gene as a node and significan co-expression relationship as an edge.

Explore co-expression gene sets in tumor and blood tissue. Visualize gene expression together

with clinicopathological variables associated with each module. Include results of gene set analyses that describe the underlying biological functions of the modules.

Explore relationships between modules from each tissue. Visualize how modules from each tissue are related using two different metrics, ranksum and gene overlap. Also enable subtype selection, enabling users to investigate relationships within a particular subtype.

Explore relationships between clinical variables and modules. Visualize significant associations between module expression and clinical variables.

Explore association between user-submitted gene lists and computed modules. Allow users to upload own gene lists and have the application compute modules which the gene list is enriched for.

Search for genes or gene lists of intrest. Allow users to search for specific genes or genelists of intrest and show what modules they are associated with.

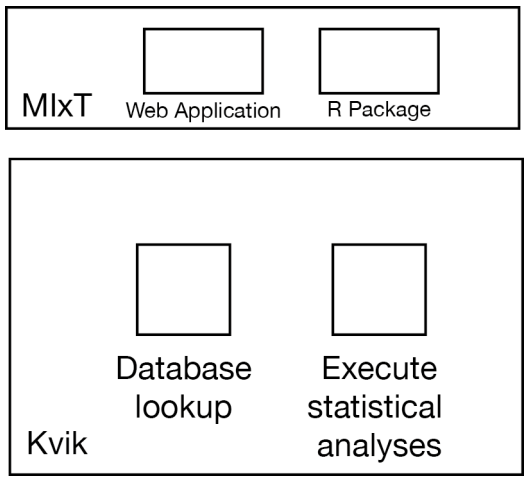

Figure 1 An overview of the relationship between the MIxT application and Kvik. MIxT contains a web application (online at mixt-blood-tumor.bci.mcgill.ca) and the R Package that provides analyses and data to the web application. Kvik provides the services for running the statistical analyses from the R package, and the database lookups found in the web application.

Design and Implementation

The MIxT application is designed as a modern application consisting of multiple services that together provide an interactive web application. By composing an application of a set of services we can substitute parts of the application without re-writing the entire application. This type of architectural style is called a microservices architecture and is popular in 'web-scale' systems. For example if we want to use OpenCPU to interface with data analysis we can do so by simply exchanging the Kvik R service with OpenCPU. Both services communicate over HTTP and their interface is the same.

From our initial analyses we built an R package with functions to provide data and analysis to the different analysis tasks. Using this design it is possible to either explore the data through the web site or a local R session.

To explore the co-expression relationship between genes we use an interactive graph visualization build with Sigmajs[5]. We have built visualization for both tissues, with graph sizes of 2705 nodes and 90 348 edges for the blood network, and 2066 nodes and 50 563 edges for the biopsy network. The sigmajs visualization library has functionality for generating a layout for large networks, but we generate this layout server-side to reduce the computational load on the client. To generate this layout we use the GGally package[6].

We have built modules for each tissue, and to explore gene sets associated with genes in these modules, we provide module overview pages that show gene expression visualized together with clinicopathological variables and gene set analyses that describe the underlying functions of the module.

We have used different metrics to link the modules from each tissue, ranksum and gene overlap. To visualize the associations we use the d3[7] library to build an interactive heatmap visualization.

To allow users to explore the relationship between clinical variables and the computed modules, we built an interctive heatmap visualization that visualizes the association between different metrics and each module.

Conclusions

List of abbreviations used

Competing interests

The authors declare that they have no competing interests.

Author details

[1]Department of Computer Science, UiT – The Arctic University of Tromsø, 9037 Tromsø, NO. [2]University of ...., , . [3]Department of Community Medicine, UiT – The Arctic University of Tromsø, 9037 Tromsø, NO.

References

[5] sigmajs.org

[6] cran.r-project.org/web/packages/GGally

[7] d3js.org