

Chapter 5

MGG: A customizable software infrastructure for genetical genomics

Morris A Swertz, Bruno M Tesson, Richard A Scheltema, Gonzalo Vera, Ritsert C Jansen

Groningen Bioinformatics Centre, Groningen Biomolecular Sciences and Biotechnology Institute, University of Groningen, The Netherlands

Submitted

Abstract

Motivation | A common software infrastructure for the growing number of genetical genomics experiments would ease sharing of data and analysis tools within and across studies on human, mouse, *A. thaliana*, *C. elegans* and other organisms. This common infrastructure is still left wanted.

Results | We report on MGG, a highly customizable software infrastructure for genetical genomics data, with a graphical user interface for biologists and programmatic interfaces for bioinformaticians. A tailor-made variant of MGG suiting your experiments can be generated using the MOLGENIS generator. With this initiative, genetical genomics groups can stay on the same track: they can share data and software tools notwithstanding large variation between research aims. This greatly increases utility for biological research, while costs and development time are much reduced.

Availability | Download MGG, or generate your own customized variant, at <http://www.molgenis.org/variants/mgg>.

About Chapter 5

This chapter reports the second of four case-studies. This purpose of this case was to refine the 'generative strategy' (**Chapter 2**) for software infrastructures to support the 'dry' lab: to (i) deal with the diversity of resources (data, algorithms, tools) such that we ease reuse and integration where possible, and specialization when needed and (ii) to handle the large dataset uploads that have to be parsed into their basic information elements to enable querying (in contrast to 'black box' in the wet lab, see **Chapter 4**).

1. Introduction	
METHODS	CASES
2. Problem analysis and approach	4. Infrastructure for the wet-lab
3. Generative development in action	5. Infrastructure for the dry-lab
	6. Infrastructure for clinical trials
	7. Reusable assets for processing
8. Discussion and Future work	

5.1 Introduction

The strategy of genetical genomics (Jansen and Nap, 2001) has been applied to the genetic study of gene expression data in a wide range of organisms including human, yeast, mouse, rat, *C. elegans* and *A. thaliana*. See Rockman and Kruglyak (2006) for a recent review. The genetic study of gene expression data can also be combined with the genetic study of other types of biomolecular data. A recent study used Qiagen-Operon microarrays to observe gene expression (Keurentjes, et al., 2007) and high-resolution LC-MS to observe metabolite abundance (Keurentjes, et al., 2006). They treated gene expression and metabolite abundance data over all individuals as quantitative traits, and used quantitative trait locus (QTL) mapping to identify expression QTL (eQTL) and metabolite QTL (mQTL) and to reconstruct regulatory and metabolic networks.

Some computational tools for the genetic study of biomolecular data have been made available (Carey, et al., 2007; Fu, et al., 2007; Li and Burmeister, 2005; Mueller, et al., 2006). These tools are typically implemented using the open source statistical R software. Computational tools and data storage are integrated in GeneNetwork.org (Chesler, et al., 2005), a first and successful software infrastructure for genetical genomics data from the mouse community. However, GeneNetwork.org is designed neither to evolve substantially over time nor to be customized to other organisms and types of biomolecular data: that would require many minor, and sometimes major, changes in hand-written software code.

Here we describe MGG: a highly customizable software infrastructure for genetical genomics. MGG thanks its unprecedented customizability to the strategy of ‘generative software development’ recently introduced by several bioinformatics projects. See Swertz and Jansen (2007) for a recent review. MGG stores *assays*, *traits*, *subjects* and *data* in four core data types and has systematic extension mechanisms to add custom variants thereof. Biologists can use a graphical user interface (GUI) to add and find back data, and bioinformaticists can use application programming interfaces (API) in R, Java and SOAP to connect MGG to processing tools. The MOLGENIS generator (Swertz, et al., 2004; Swertz and Jansen, 2007) is used to quickly produce customized versions of MGG. The common data model and uniformly generated interfaces help sharing of data, customizations and processing tools, notwithstanding large variation between research aims. We invite genetical genomics researchers at www.molgenis.org/variants/mgg to view the demo, download, use and extend MGG.

5.2 Data model

Figure 1 shows MGG’s four core data types: *Subject*, *Assay*, *Trait*, and *Data*. In short, *subjects* describe the biological samples mousestudied, *assays* describe the biomolecular and/or computational protocols used, *traits* are the biological and/or computational features in the assays, and *data* are the observations/calculations for traits produced by the assay.

Variants of core data types can be added. In fact, **Figure 1** shows *Subject* and *Trait* to point to an additional core data type called *Item*. The solid triangled line indicates that *Subject* and *Trait* (line) are variants of *Item* (triangle) ‘inheriting’ all its properties. If each *Item* has a name, an identification number and a type, then each *Subject* and *Trait* also has a Name, ID and Type. **Table 1** illustrates the usage of these data types.

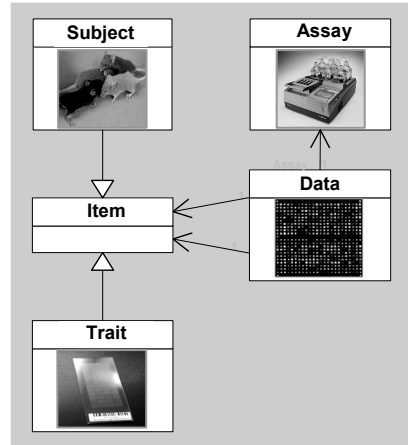


Table 1 | Use cases of core data types

Figure 1 | Core data types.

A growth measurement (<i>Assay</i>) reports the time (<i>Data</i>) it took to flower (<i>Trait</i>) for an Arabidopsis plant (<i>Subject</i>).
A two-color microarray measurement (<i>Assay</i>) reports a list of raw intensities (<i>Data</i>) of hybridized gene expression probes (<i>Trait</i>) for one pair of Arabidopsis individuals (<i>Subject</i>).
A marker based genetic profiling (<i>Assay</i>) reports genotype values (<i>Data</i>) for each SNP/microsatellite marker (<i>Trait</i>) for one mouse individual (<i>Subject</i>).
A kinome array measurement (<i>Assay</i>) reports the activity (<i>Data</i>) of certain enzymes (<i>Trait</i>) in a mouse individual (<i>Subject</i>).
An LC/MS time series (set of <i>Assay</i>) reports the intensity (<i>Data</i>) of certain mass/charge peaks (<i>Trait</i>), indicating metabolite abundance for a bowel sample on human (<i>Subject</i>).
A peak identification algorithm (<i>Assay</i>) reports the identification (<i>Data</i>) of certain mass peaks (<i>Trait</i>) to be a certain metabolite (<i>Trait</i>).
Correlation computation (<i>Assay</i>) reports associations (<i>Data</i>) between metabolites (pairs of <i>Trait</i>).
A QTL mapping (<i>Assay</i>) reports all p-values (<i>Data</i>) mapping genes (<i>Trait</i>) to marker positions (<i>Trait</i>) for recombinant inbred lines of <i>C. elegans</i> (<i>Subject</i>); this assay is derived from microarray and genetic profiling assays (follow derivedFrom relationship in Figure 2).

Extended variants of *Subject* and *Trait* are shown in **Figure 2**. A *MassPeak* inherits from *Trait* and has properties ID, Name and Type (like *Trait*), but also additional properties Mass, RetentionTime and Intensity. A *Marker* also inherits from *Trait* and adds properties (StartBP, EndBP, cMPosition, etc) that specify its genomic and genetic location, but here these have been combined in an ‘interface’ called *Locus*. Other data types can also ‘implement’ the interface to *Locus* to get the same additional properties (e.g. a *Probe* also has StartBP, EndBP, cMPosition, etc). Thus both a *Marker* and a *Gene* have in common that they ‘behave like’ a *Locus*. **Table 2** shows examples of data type variants.

Table 2 | Use cases of data type variants

A Strain is a Subject.
Sample is a Subject with the additional property that 'Tissue' has to be specified.
Individual is a Subject with the additional property that relationships with Mother and Father individuals, as well as Strain, can be specified (arrows).
TwoColorSample is a Sample with the additional property that 'Dye' has to be specified and which two Subjects (or subclasses like Individual) are labeled with 'Cy3' and 'Cy5' (arrows).
An InbredStrain is a Strain with the additional property that the 'Parents' (mother Individual and father Individual) are specified and the 'type' of inbreeding used.
An AFLP, micro satellite or SNP Marker (is a Trait) may refer to genetic and possible genomics location (Marker behaves as Locus).
A genetic map (Assay) shows combinations (Data) of genes (Gene is a Trait) and Marker positions (Marker behaves as Locus and is a Trait) for a cultivar (Strain) of tulip (Species).

Figure 2C shows three more data types *Experiment*, *Reporter* and *Protocol*. In short, *experiments* have a name and summary and bind all traits and/or subjects and assays together; *reporters* are artificial traits that report for a real trait (similar to MAGE reporters, see Spellman, et al., 2002); and *protocols* describe assays in a central place, optionally including lists of reporters to be used. **Table 3** illustrates the usage of these three data types.

Table 3 | Use cases of data types Experiment, Protocol and Reporter

A Probe (<i>Reporter</i>) is a <i>Trait</i> that reports for the expression of <i>Genes</i> ; as <i>Reporter</i> it has the additional property 'reportsFor', in this case, a <i>Gene</i> (arrow).
The Affymetrix 'Mouse Genome 430 2.0' microarray <i>Protocol</i> contains 495,374 <i>Probes</i> (<i>Reporters</i>) organized in 45,037 <i>ProbeSets</i> that report for 39,000 <i>Genes</i> (<i>Traits</i>).
A genetical genomics <i>Experiment</i> on Stem Cells was carried out on 30 recombinant InbredStrains (<i>Subjects</i>). It used the 'Affymetrix MG-U74Av2' <i>Protocol</i> to produce expression profiles (<i>Assay</i>), having 12,422*16 microarray <i>Probes</i> (<i>Traits</i>). The result included a matrix of signals (<i>Data</i>) for each Probe (<i>Traits</i>) and each InbredStrain (<i>Subject</i>).
A genetical genomics Arabidopsis thaliana flowering <i>Experiment</i> contains genetic profiles (<i>Assay</i>) with <i>Data</i> on 144 <i>Markers</i> (<i>Trait</i>) and 160 Arabidopsis recombinant (RI) <i>InbredStrain</i> (<i>Subject</i>). Also unidentified metabolite abundance profiles (<i>Assay</i>) are available with <i>Data</i> on 2,129 <i>MassPeak</i> (<i>Trait</i>) and 160 <i>InbredStrains</i> , and QTL profiles (<i>Assay</i>) with <i>Data</i> describing the mapping of 2,475 <i>MassPeak</i> on 144 <i>Markers</i> .

Not shown in **Figure 2** are the five data types *BibliographicReferences*, *DatabaseEntry*, *OntologyTerm*, *URI*, and *FileAttachement*, see Pizarro et al (2006) instead. These data types enable annotation of the four data types *Item*, *Assay*, *Experiment* and *Protocol* in a flexible way. For example, a *Gene* can have more than one *OntologyTerms* (but doesn't

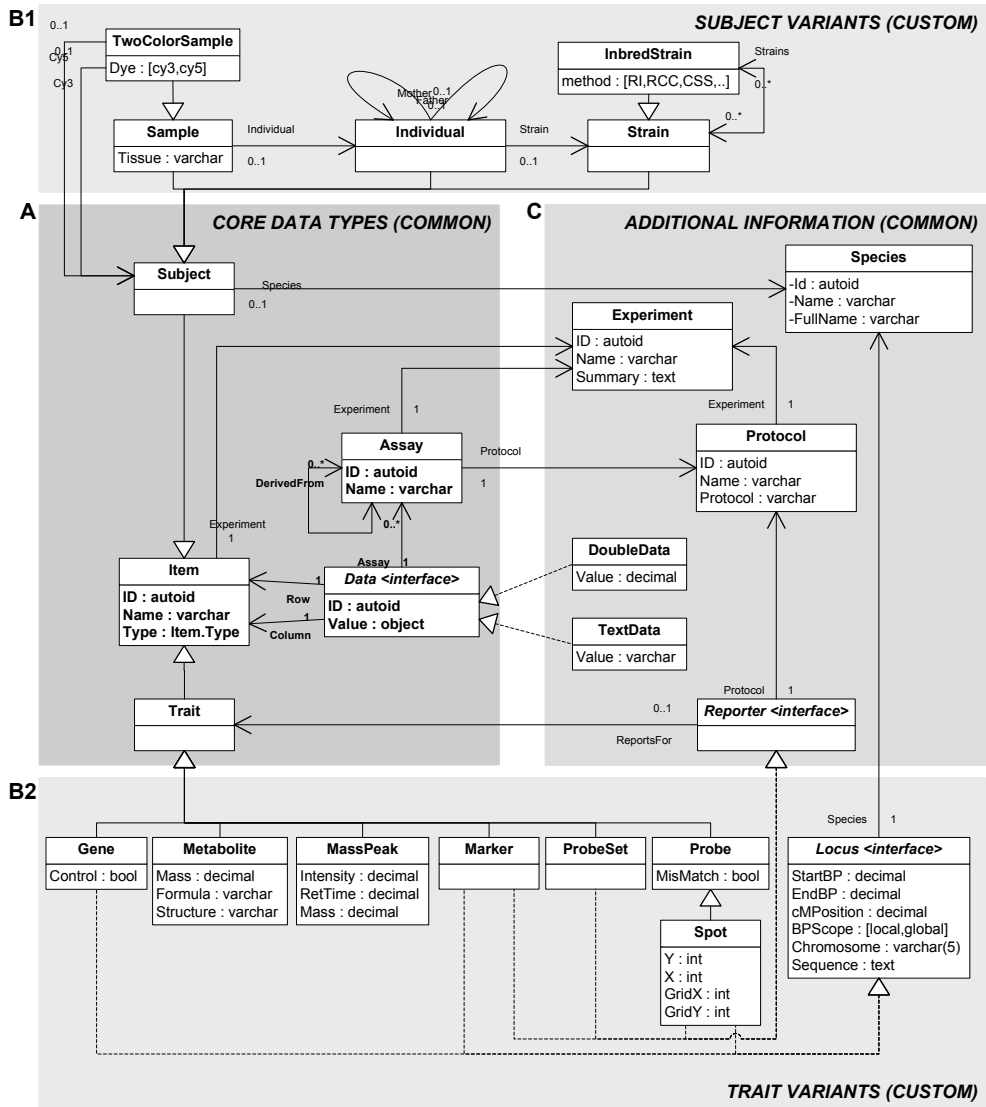


Figure 2 | **Genetical genomics data model.** Genetical genomics data can be described using core data types, e.g. *Subject* and *Trait* (A). Specific attributes and relationships can be described by adding data type variants, e.g. *Sample* and *Gene* (B). Additional information can be added, e.g. *Experiment* and *Protocol* (C). See Table 1-4 for uses of this model. Arrows denote relationships (*Item* has a field *Experiment* that refers to *Experiment* ID); Triangled lines denote inheritance (*MassPeak* inherits all properties ID, Name, Type next to Mass, Intensity and Ret(ention)Time); Triangled dotted lines denote use of <interfaces> (*Spot* ‘implements’ properties of *Locus*).

have to) and a *Metabolite* can link to zero or more *DatabaseEntry*. The drawback of this flexibility is that users and tools using *Metabolite* and *Gene* will have to inspect each instance to see whether it is annotated. That is why we used inheritance (above) for essential properties of *Trait* and *Subject* variants above to make sure that they are described in a uniform way. See Jones and Paton (2005) for a discussion on the benefits and drawbacks of alternative mechanisms to deal with data variety. **Table 4** illustrates the usage of these annotation data types.

The simple and concrete data structures described above greatly help to become more productive with the data, in line with findings of Brazma (2006) and Rayner (2006). To understand most data in MGG, newcomers just have to learn the four core data types because variants thereof follow a uniform pattern. To ensure this simplicity, we *did not* create variants of *Data* to accommodate various combinations of *Trait* and *Subject*, i.e. *ExpressionData*, *MassSpectrometryData*, *QtlMappingData*, *PeakCorrelationData*, etc. Instead, we store all data using only *Data* which greatly reduced the number of variants needed. However, we *did* create variants for *Trait* and *Subject*. This concreteness allows a programmer of a Genome Browser tool to depend on each *Probe* and *Marker* describing ‘genomic location’ in a common way. This greatly reduces development effort and eases the sharing of tools between MGG instances.

Table 4 | Use cases of annotation data types

A *Gene* in an *Arabidopsis Experiment* can be connected to a *DatabaseEntry* describing a reference to related information in the TAIR database (<http://www.arabidopsis.org>) and another *DatabaseEntry* describing a reference to the MIPS database (<http://mips.gsf.de>).

For each *Probe* a *URI* was added that describes a hyperlink to GeneNetwork.org.

The *Arabidopsis Experiment* was annotated with the *BibliographicReferences* pointing to the paper describing the experiment and expected results.

A *Protocol* describes the ‘MapTwoPart’ method for QTL mapping and was annotated with the *URI* linking to the ‘MetaNetwork R-package’, which contains this method, and an *BibliographicReference* pointing to the paper (Fu, et al., 2007) which describes the MapTwoPart protocol.

A file with a Venn diagram describing the number of masses detected in each RIL population was added as *FileAttachment* to the *Arabidopsis metabolite Experiment*.

A list of *Genes* was selected from a trans band in QTL profiles (*Assay*) and annotated with Gene Ontology terms (*OntologyTerm*) and relevant literature annotations (*BibliographicReference*).

5.3 Graphical user interface

Figure 3 shows the graphical user interface (GUI) to upload, manage, find and download genetical genomics data in the database. The GUI is generated in a uniform way lowering the barrier for novel users. Experiments can be described with all subjects and traits

involved¹. Data can be entered using either the edit boxes or using menu-option ‘file|upload’². This option enables upload of whole lists of traits and subjects from a simple tab-delimited format³, which can easily be produced with Excel or R; MOLGENIS automatically generates online documentation describing the expected format⁴. Subsequently, assays involved can be added with the resulting raw data (e.g. genetic fingerprints, expression profiles) and processed data (e.g. normalized profiles, QTL profiles). These data can be uploaded, again using the common tab-delimited format or custom parsers⁵ that bioinformaticists can ‘plug-in’ for specific file formats (e.g. Affymetrix CDF and CEL files). The GUI checks the relationships between data, traits, and subjects so no ‘orphaned’ data is loaded into the database, e.g. genetic fingerprint data cannot be added before all information is uploaded on the markers and subjects involved.

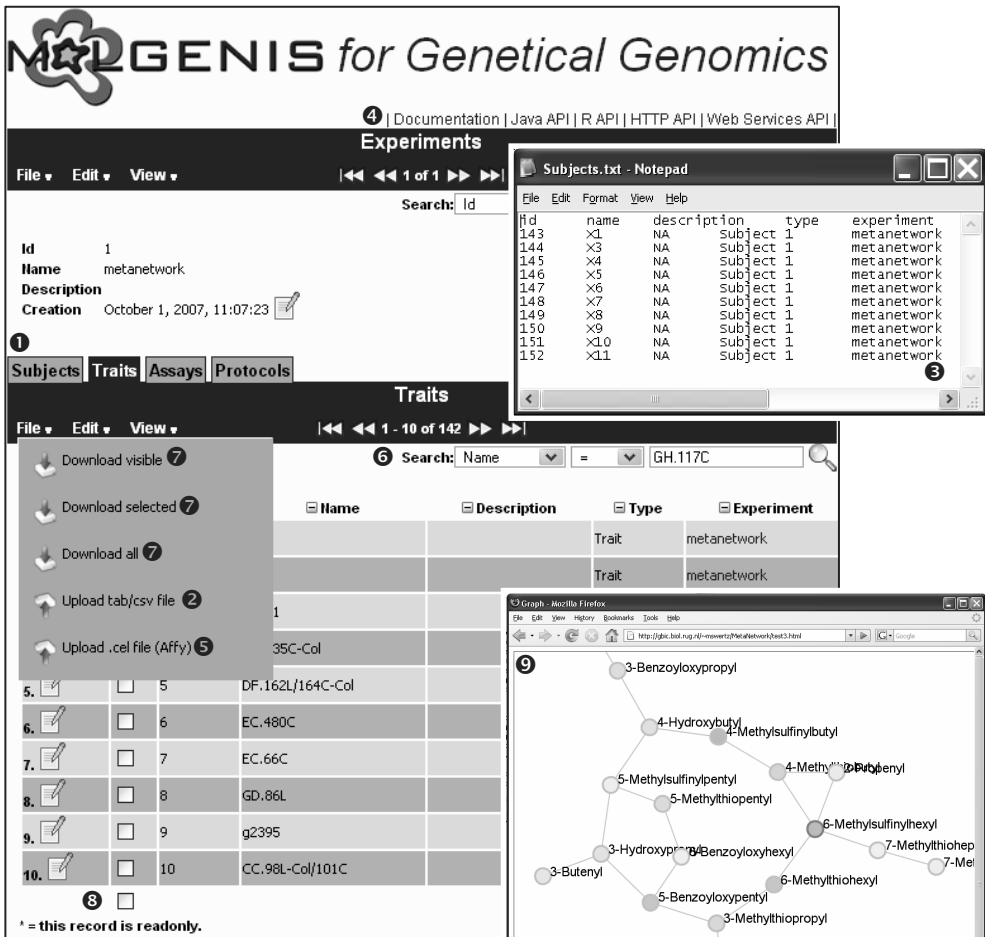


Figure 3 | **Graphical user interface.** Genetical genomics information can be explored by experiment, items, traits or assay. Hyperlinks following cross-references of the data model point to related information. Items indicated by ¹⁻⁹ are described in the main text. See also online demo: www.molgenis.org/variants/mgg

Biologists can use the graphical user interface to navigate and retrieve available data for analysis. Optionally, public access to experiment data can be regulated by assigning edit/read rights and user roles (not shown). They can use the advanced search options ⑥ to find certain traits, subjects, or data. Using menu option ‘file|download’ ⑦ they can download visible/selected ⑧ data as tab-delimited files to analyze them in 3rd party software. Bioinformaticians can ‘plug-in’ a custom build screen (see ‘customization’ section) that allows processing of selected data inside the GUI, e.g. visualize a correlation matrix as a graph ⑨ without the additional steps of downloading data and uploading it into another tool. Biologists can follow links-out to related information, for example to probes in GeneNetwork.org (not shown). **Table 5** summarizes use cases of the graphical user interface:

Table 5 | Use cases of the GUI for biologists

Navigate all <i>Experiments</i> , and for each <i>Experiment</i> , see the <i>Assays</i> and available <i>Data</i> .
Select a <i>Gene</i> and find all <i>Experiments</i> in which this <i>Gene</i> shows significant eQTL <i>Data</i> (p-value < 0.001).
For a given <i>Locus</i> , select all <i>Genes</i> that have QTL <i>Data</i> mapping ‘in trans’ to this <i>Locus</i> , e.g. absolute(QTL locus – gene locus) > 10Mb and QTL p-value < 0.001.
Download a selection of raw gene expression <i>Data</i> as tab-delimited file (to import into other software).
Upload <i>Experiment</i> information from tab-delimited files.
Upload Affymetrix <i>Assays</i> using custom CEL/CDF file readers.
Plot highly correlated mQTL <i>Data</i> in a network visualization graph.
Define security levels for <i>Assays/Experiments</i> to ensure that appropriate data can be viewed by collaborators, and not by other people.
A <i>MassPeak</i> has been identified to be ‘proline’ and we can follow the link-out <i>URI</i> to Pubchem (http://pubchem.ncbi.nlm.nih.gov/), because it was annotated to have ‘cid’ 614, to find information on structure, activity, toxicology, and more.

5.4 Application programming interfaces

Bioinformaticians can connect their particular R or Java programs to genetical genomics data in MGG using an application programming interface with similar functionality as the graphical user interface (R/API, J/API). Scripts in other programming languages can use web services (WS/API) or a simple hyperlink-based interface (HTTP/API). When MGG is customized with additional data type variants, the API is automatically extended by the MOLGENIS generator allowing interaction with these new types in the uniform way.

Figure 4A-E shows how one can use the R/API to upload all trait/subject/datasets involved (**A**). Another researcher can download these data into statistical scripts to calculate QTL profiles (**B**) to find information on metabolites that significantly map to the genomic

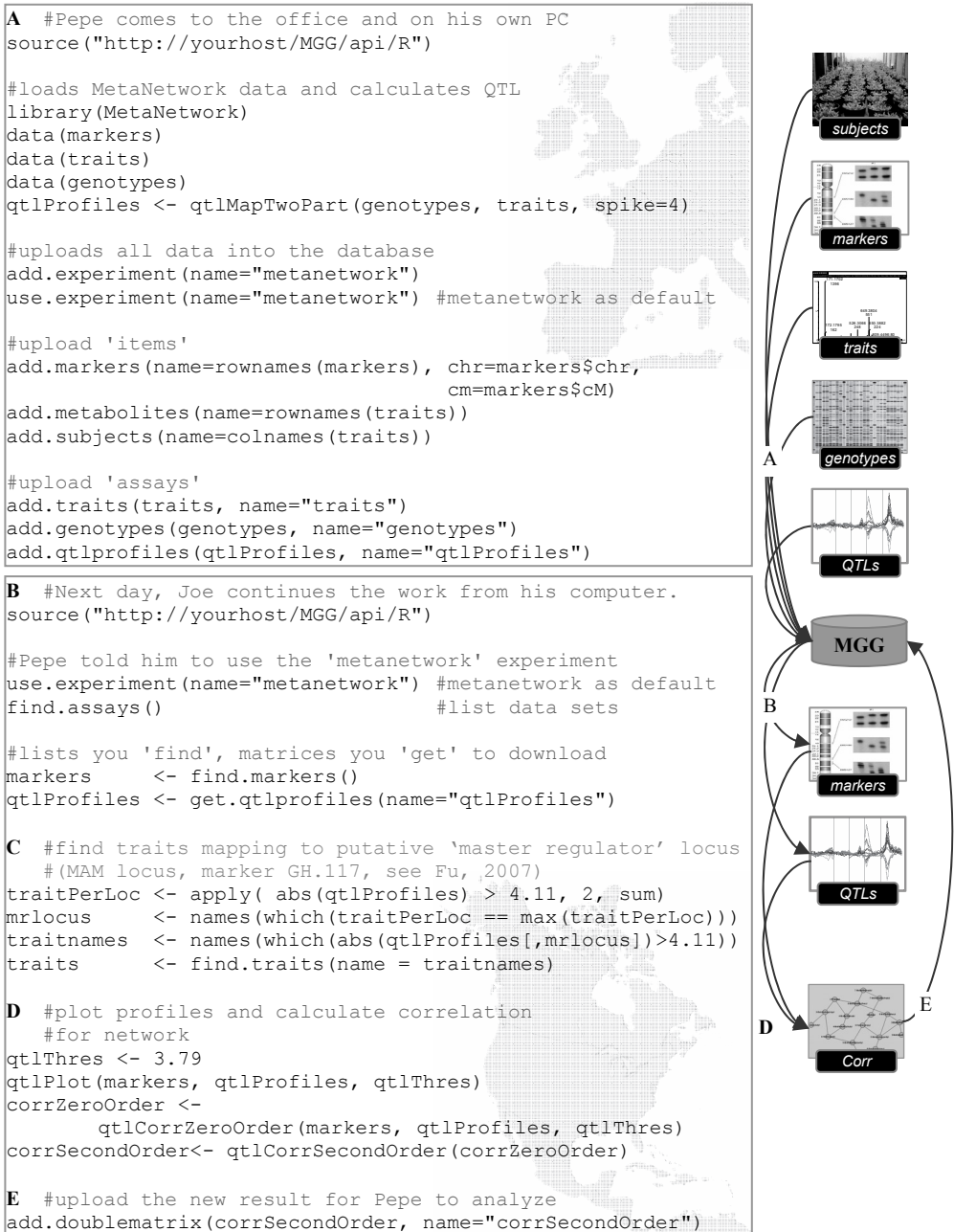


Figure 4 | **Application Programming Interface.** Genetical genomics information in MGG can be explored programmatically, either using Java, R, SOAP or HTTP request. The figure shows use of the R/API to upload data (A), to download data for calculation of QTL profiles (B), to find traits putatively controlled by a master regulator locus (C), to plot profiles and calculate correlation for network reconstruction (D), and to upload the results thereof to the database so it can be used by another collaborator (E).

location with the most significant QTLs suggesting control by a master regulator (**C**), and to calculate correlation networks (**D**), building on genetical genomics R packages like MetaNetwork (Fu, et al., 2007). The results thereof can be uploaded again for exploration by others (**E**). The GUI and API are structured similarly so that a researcher ‘talking’ to MGG with the API can trace what has happened using the GUI. Click the R/API, J/API, WS/API and HTTP/API buttons in the GUI to find documentation on using the APIs.

Upload of an experiment requires integration of many data files providing information on assays, traits, subjects and data. The API helps a lot to describe all the necessary steps to add data (see **Figure 4**). The entries in these files should be correctly connected, but in practice these connections take some effort to ‘reconstruct’. The data must be imported in the right order and the names of gene ids should be identical between all the files describing genes, microarray probes and gene expression. With the APIs an import can be quickly defined, including (id) conversions if needed, and then automatically run for the import of every next data set. Moreover, the J/API supports ‘transactions’ which ensures that all data inserts are rolled back when the import fails halfway preventing incomplete data. Therefore, we preferred using the API over using file-uploads in the GUI for importing genetical genomics datasets on a regular basis. **Table 6** summarizes use cases of the application programming interface.

Table 6 | Use cases of the API for bioinformatician

In R, parse a set of tab-delimited <i>Marker</i> , <i>Genotype</i> and <i>Trait</i> files and load them into the database (R/API).
In R, retrieve all <i>Traits</i> , <i>Markers</i> , expression <i>Data</i> , and genotype <i>Data</i> from an experiment as data matrices, before QTL mapping with MetaNetwork (R/API).
In Java, retrieve a list of QTL profile correlation <i>Data</i> to show them as a regulatory network graph (J/API).
In Java, customize generated file readers to load specific file formats (J/API).
In Taverna, retrieve <i>Genes</i> for MGG to find pathway information in KEGG (WS/API).
In Python, retrieve a list of QTL mapping <i>Data</i> using a hyperlink to MGG (HTTP/API).

5.5 Customizing MGG

We used MOLGENIS (Swertz, et al., 2004; Swertz and Jansen, 2007) to allow quick customization and extension of the software infrastructure. On the push of a button, the MOLGENIS generator automatically produces a ready-to-use software infrastructure with GUI and APIs from a file written in a simple *domain specific language* (DSL).

Figure 5A demonstrates how we used MOLGENIS’ DSL to describe the core data types *Trait* and *Data* as entities; **Figure 5B-F** demonstrates how to extend MGG; re-running the generator on an updated DSL file will instantly produce an extended version of the software infrastructure. **Figure 5B** shows how the addition of *Gene* and *Marker* as new variants of

Trait takes only a few additional lines. **Figure 5D** shows how to create a ‘matrix’ data type, especially useful inside R (e.g. a Genotype matrix that is stored in *TextData* and has *Markers* as rows, and *Subjects* as columns). **Figure 5E** shows how to create ‘dynamic’ hyperlinks to other websites (e.g. to TAIR, Entrez or Ensembl). **Figure 5F** programmers can add a ‘plug-in’ program that is not generated by MOLGENIS but written by hand in Java (e.g., a viewer that plots qtl profiles interactively).

The generator can be run either online or offline, see www.molgenis.org/tools. MOLGENIS either generates a ‘server’ edition (which requires installation on server

```

A
<entity name="Trait" extends="Item">
  <field name="Id" type="autoid"/>
  <field name="Name" type="varchar" length="128"/>
  <field name="Description" type="text" nillable="true"/>
  <field name="Project" type="xref" xref_field="Project.Id"
    xref_label="Name"/>

  <unique fields="Id"/>
  <unique fields="Name,Project"/>
</entity>
<entity name="Data">
  <field name="Id" type="int" auto="true"/>
  <field name="Assay" type="xref"
    xref_field="Assay.Id" xref_label="Name"/>

  <field name="Row" type="xref"
    xref_field="Item.Id" xref_label="Name"/>

  <field name="Col" type="xref"
    xref_field="Item.Id" xref_label="Name"/>

  <field name="Value" type="object"/>
  <unique fields="Id"/>
</entity>
B
<entity name="Gene" extends="Trait" implements="Locus">
  <field name="Control" type="bool"/>
</entity>
<entity name="Marker" extends="Trait" implements="Locus"/>
C
<entity name="Locus" interface="true">...
D
<matrix name="Genotype" content_entity="TextData"
  container_field="Assay" content_field="Value"
  row_field="Row" row_entity="Marker"
  col_field="Col" col_entity="Subject">
E
  <field type="hyperlink" pattern="webqtl?cmd=get&probe=${ID}"/>
F
  <plugin name="myplugin" entity="Data" class="NetworkViewer"/>

```

Figure 5 | **Generating a custom variant of MGG.** A file in MOLGENIS domain specific language is used to generate the database infrastructure (see **Figure 5**). (A) Properties of core data types *Trait* and *Data* are described as ‘fields’. (B) Specific variants of core data types can be created in a few lines using ‘extends’. (C) Properties common to multiple data types can be defined once as ‘abstract’ interface to be reused many times using ‘implements’. (D) Users can interact with database tables as if they were a matrix. Hand-written extensions can be added to the generated infrastructure using a (E) dynamic hyperlink or a (F) ‘plug-in’. See <http://www.molgenis.org/variant/mgg> for the complete DSL file that also describes the GUI.

software) or a ‘standalone’ edition that runs on your computer (at the click of a mouse-button). This choice can be made via a checkbox (online) or configuration file shows how ‘molgenis.properties’ (offline). The complete DSL file and generated source-code are provided at <http://www.molgenis.org/variants/mgg> under the open source GPL. The MOLGENIS generator and documentation are available online at <http://www.molgenis.org/> under a dual license, either LGPL for universities or a commercial license for companies.

5.6 Technical details

Figure 6 summarizes how MOLGENIS generates MGG in three layers: a database (DB), an application programming interface that builds on the database (API), and a graphical user interface that builds on the API (GUI). Below we summarize the implementation details; a manuscript is in preparation providing more detail on MOLGENIS (Swertz, et al., Submitted).

Database

MOLGENIS generates a SQL file with ‘database CREATE statements’ that is loaded into either HSQLDB or MySQL. Each data type in the data model (see **Figure 2**) is mapped to

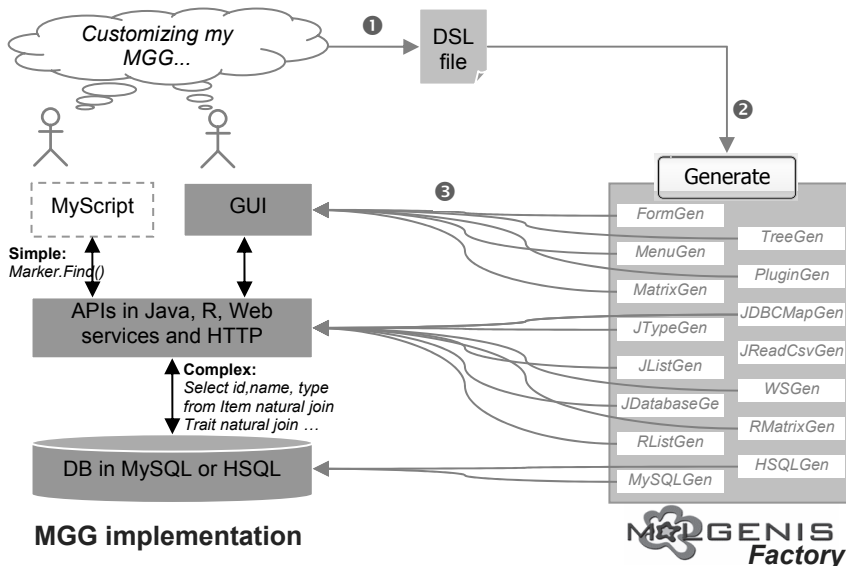


Figure 6 | Implementation and customization. ❶ The MGG software infrastructure is described using MOLGENIS domain specific language (DSL, see **Figure 5**). ❷ Central software termed MolgenisFactory runs several generators, building on MOLGENIS catalogue of reusable assets. ❸ On the push of the ‘generate’ button, all the software code for a working MGG implementation is automatically generated from the DSL file. GUI and APIs provide simple tools to add and find back data, while the reusable assets of MOLGENIS hide the complexity normally needed to implement such tools. For customization, simply change the DSL file and the MOLGENIS generator takes care of rewriting all the necessary files of software code.

its own table, e.g. there is a table ‘Trait’. Each inheritance adds another table, e.g. each *Gene* has an entry in table ‘Gene’ and also in table ‘Trait’. One-to-many cross-references between data types are mapped as foreign keys, e.g. *Data* has a numeric field called ‘assay’ that must refer to the *foreign key* ‘id’ of *Assay*. Many-to-many cross-references are mapped via a ‘link-table’, e.g. an additional table *AssayToAssay* is generated with two foreign keys, both to *Assay* id, to record ‘derivedFrom’ connections between two *Assays*.

API

Programmers can use the simple commands like ‘find’, ‘add’ and ‘update’ via application programming interfaces in Java, R, HTTP hyperlinks and Web Services (J/API, R/API, HTTP/API and WS/API). These APIs remove the complexity of SQL database commands and also check whether incoming data conforms to the data model (e.g. from tab-delimited files, http requests, etc).

The J/API is implemented using industry standard JDBC components for database access. Javadoc-style documentation of the Java API is available via the GUI. MOLGENIS generates several optimizations to accommodate larger data sets (>100MB). First, the J/API uses ‘batched’ updates of JDBC and the ‘multi-row-syntax’ of MySQL to allow inserts of 10.000s of data entries in a single-command. This method is 5-15 times faster than inserting data entries one-by-one. Second, efficient ‘data lists’ are generated for each data type, e.g. a *TraitList* needs up to four times less memory than Java-standard ArrayLists. For example, a data set of Affymetrix experiments with 60 assays of each 190.000 expressions can be kept in memory on a simple PC with 1 GB of memory.

The other APIs are built on top of the J/API. The R/API is connected to the J/API over the Internet using Servlets for uploading and downloading. This intermediate ‘HTTP/API’ works via simple hyperlinks, e.g. <http://host/mgg/api/find/Assay?experiment=1> returns all assays for experiment ‘1’. Helper functions ease the import of R matrices into the database. For example, a matrix (assay of microarray intensities) with 160 columns and 24,000 rows (representing individuals and probes) is translated into a list of 160*24,000 table rows, where each row has four columns representing matrix-row (individual), matrix-column (probe), cel-value (intensity), and the identifier of the matrix (assay), see **Figure 4A**.

Programs written in other languages can use the WS/API that shares SOAP services described in a WSDL file. The web services are produced using JAX-WS (<https://jax-ws.dev.java.net>). Alternatively, programmers working in other languages can also use the HTTP/API (use <http://host/mgg/api> for examples) or add generators for their language. For example, addition of a the Python/API generator requires (i) a *template* that describes Python code to interact with the J/API and (ii) a simple Java ‘Generator’ class that applies the template to each ‘entity’ in the DSL file to (iii) produce specific data access methods in Python for each data type. Current MOLGENIS generators are listed in **Figure 6**.

GUI

The graphical user interface (GUI) publishes the functionality of the API to biologists. For example, a ‘TraitScreen’ is generated to find, update and browse through *Traits*. This TraitScreen builds on reusable assets: FormScreen remembers the current state of the GUI (e.g. data currently viewed), FormView layouts the screen as HTML page and FormController changes the screen-state on user input (e.g. previous, next, update). Other reusable assets are MenuScreen, TreeScreen, MatrixScreen and PluginScreen, the latter to incorporate hand-written screens. The GUI is run as Java Servlet and can be run on a server (which is advised when there are multiple users) or as standalone executable that runs on any PC with Java. The server implementation consists of a WAR file which can be run on a Servlet container such as Tomcat (<http://tomcat.apache.org>), which serves the application over the Internet and facilitates access to a MySQL database. This can be configured in the molgenis.properties file (see MOLGENIS installation manual at <http://www.molgenis.org>). The standalone implementation consists of a self-contained, executable JAR-file that is build with the HSQL database and Jetty web server (<http://www.mortbay.org>) embedded.

5.7 Discussion

In this paper we reported on MGG, a ‘Molecular Genetics Information System’ (MOLGENIS) for ‘Genetical Genomics’. **Box 1** summarizes the main features of MGG. Use of the MOLGENIS generator tool speeds up customization and ensures that the API/GUI of customized MGG systems still behave in a uniform way. This allows the community of genetical genomics research groups to use MGG to share data and tools, because different installations of the software infrastructure use uniform data, API and GUI structures notwithstanding large variation between research aims.

Groups that already have an infrastructure (e.g. GeneNetwork.org) can assimilate MGG to ease evolution of their existing software. After ‘rewiring’ of algorithms and visual tools to use the MOLGENIS API’s to manage data, researchers still have the same features as before plus the features provided by the generated infrastructure (e.g. data management, R/API) and connected tools (e.g. R packages developed elsewhere). Moreover, much less software code needs to be maintained by hand when replacing hand-written parts by MOLGENIS generated parts, allowing software engineers to add new features for researchers faster. We invite the genetical genomics community to join in on <http://www.molgenis.org/variants/mgg> to share the best customizations and plug-ins of MGG and to push developments into directions that most benefit genetics research.

Acknowledgements

We thank Jingyuan Fu, Rudi Alberts, Yang Li, and Alrik Lubbers for their valuable input in providing requirements, examples and feedback. This work was supported by grants from the Netherlands Organization for Scientific Research, Program Earth and Life Sciences, and the Netherlands Ministry of Economic Affairs, Programs Biopartner and Biorange.

Box 1 | Features of MGG – software infrastructure for genetical genomics.

Store each genetical genomics experiment using only four 'core' data types: *Trait*, *Subject*, *Assay*, and *Data*. For example: a single-channel microarray *Assay* reports raw gene expression *Data* for each microarray probe *Trait* and each individual *Subject*.

Customize MGG with extended variants of *Trait* and *Subject*. In my MGG, *Probe* traits have a sequence and genome location and *Strain* subjects have parent strains and (in)breeding method. Describe extensions in MOLGENIS language and the generator automatically changes the software.

Upload data from measurement devices, public databases, collaborating MGGs, or a public MGG repository with community data. Simply download trait information as files from one MGG and upload it into another; this works because of the uniformity of the core data types (and extensions thereof).

Search genetical genomics data using the graphical interface with advanced query tools. The uniformity of the 'code generated' interfaces make it easy to learn and use interfaces for both 'core' data types as well as customized extensions.

Analyze data by connecting tools using simple methods in Java, R, Web Services or Internet hyperlinks. For example: map and plot quantitative trait loci in R using MGG data retrieved via

```
source(http://host/mgg/api/R);  
find.assays(); get.genotypes();  
get.expressions();
```

Plug the best analysis tools into the user interface so biologists can use them. Bioinformaticists are provided with simple mechanisms to seamlessly add such tools to MGG, building on the automatically generated GUI and API building blocks.

Share data, customizations, connected analysis tools and user interface plug-ins with the genetical genomics community, using MGG as exchange platform. For example, the MetaNetwork R package can talk to data inside MGG. This makes it easy for other MGG owners to also use it.