Application Coupling and Data Feeding in an Integrative Life Sciences Data Management System

***Abstract*—Life sciences research aims at studying complex interactions in biological systems in an integrative way. To link together inherently distributed data and applications generating this data, a data management solution is required that i) allows the integration of large amounts of data generated by instruments and analysis applications and ii) supports the ad-hoc loosely coupling of external applications with a common data management infrastructure. Then, dynamic data exchange between the internal and external data repository should be possible which includes feeding with external data by any application. Users of such a system should be able to assemble appropriate input data from the logically integrated data repository and run external applications provided by developers. In this paper, we present how we extended our B-Fabric life sciences data management system to support ad-hoc coupling and dynamic feeding of applications. We show how this can be realized on-the-fly with small expenses for the developer. We also sketch a mechanism how external applications can feed the B-Fabric data repository.**

I. Introduction

In recent years life sciences research aims more and more at characterizing complex biological organisms and functions at the systems level [15]. To achieve this ambitious goal, data produced in different research projects and groups must be linked together. To match the complexity of the analytical systems in systems biology research and to combine data from the different research fields, a system for integrated management of experimental data and scientific annotations is needed. Some help regarding data management is provided with the use of a LIMS (Laboratory Information Management System). A review of available systems with regard to proteomics research can be found in [10]. In contrast to the usefulness of such systems, specific isolated application-oriented data management and analysis has become apparently inefficient and a large part of generated data is lost for further analysis. The goal of a general data management and analysis system would be to capture and integrate all generated scientific data in a way that also third parties can correctly interpret, reproduce, and use the data in their own scientific context plus to provide features for the analysis steps themselves.

In [26] we motivated the need for such a system in more detail and introduced B-Fabric, an open source life sciences data management system implemented and deployed at the Functional Genomics Center Zurich (FGCZ). B-Fabric is now running for two years at FGCZ and is continuously used and extended with new features. Basically, it allows to store and annotate all data produced at FGCZ and offers a structured way of data retrieval for the user. Besides internal storage capacity, any external data store can be attached and made accessible via B-Fabric. Users do not need to care about where and how the data are kept. B-Fabric captures and provides the data transparently and in access-controlled fashion through a Web portal. Using its search and browse features, inter-experiment and inter-project analysis becomes possible. Since experimental data is captured together with annotations like instrument and processing parameters, experiments become reproducible for third parties.

In this paper, we present a novel feature of B-Fabric which allows to dynamically couple external applications with the system, whether be commercial packages or in-house developments. Once an application is registered with B-Fabric, users may invoke and feed the application via B-Fabric. Through application registration, the functionality of B-Fabric can be extended at run-time without changing the core code base. Research centers like the FGCZ that have both researchers with bioinformatics background and users without profound computer knowledge can benefit from such an infrastructure since it provides an easy way for the users to invoke and feed applications and scripts from the bioinformaticians with data from the B-Fabric repository. The bioinformaticians can meanwhile continue their usual programming work, creating small programs and scripts to solve specific research questions, and once an application is ready, plug it into the B-Fabric system and announce it instantly. A major advantage of this approach is that the software development cycle of B-Fabric is decoupled from the individual development pace of the bioinformaticians. New applications can therefore go online prior to B-Fabric release dates. Also, the applications can be written in any programming language and do not need to be altered for the coupling with B-Fabric. Examples for such applications are visualization of mass spectrometry spectra, conversion tasks, or Gene Ontology enrichment analysis of expression data.

The rest of the paper is organized as follows: Section II gives a brief overview of the B-Fabric architecture and sketches how data and application integration works in B-Fabric. Section III presents in detail the B-Fabric approach to register and invoke (external) applications. Section IV explains the process of feeding application results back into B-Fabric. Section V concludes the paper.

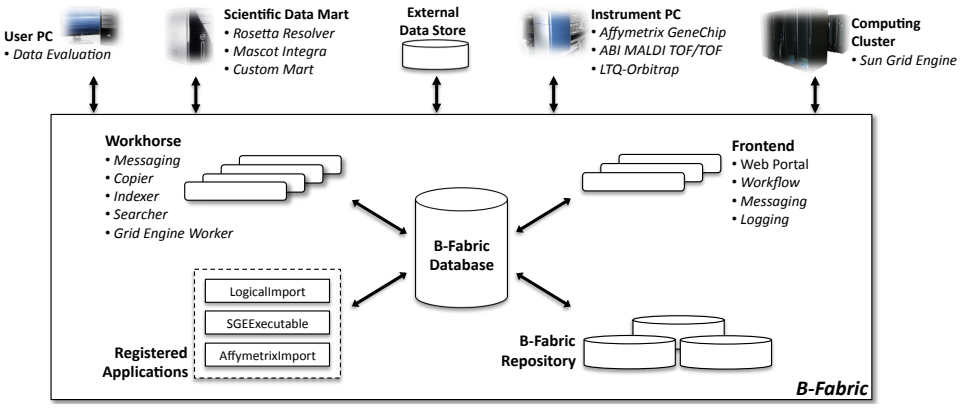

Fig. 1. B-Fabric System Architecture

II. B-Fabric

A. System Architecture

B-Fabric is an open source data management system for life sciences applications [5]. It is composed of distributed and loosely-coupled internal components based on open source technologies as well as external components which allow B-Fabric to extend dynamically. Figure 1 shows an overview of the B-Fabric architecture. Internal B-Fabric components are depicted within the big box labeled as B-Fabric. The external components reside outside the box.

There are five main internal components of B-Fabric. At the core of B-Fabric there exists *(i)* the *B-Fabric Database* which stores and manages all scientific annotations and base administrative data, e.g. about projects and users. B-Fabric currently deploys a *PostgreSQL* [18] database for this component. Instead of the latter, any off-the-shelf SQL database management system could be used as the *B-Fabric Database*, provided that it has the *Apache OJB* [4] support for object-relational mapping from Java to SQL. All experiment (raw and processed) data are stored in *(ii)* the *B-Fabric Repository*. It is a *RAID-6*-based storage that provides high reliability. In order to maintain the consistency of data, low level access to the repository is forbidden. The repository is only accessible through B-Fabric. B-Fabric provides users with *(iii) Frontends* which are accessible via Web browsers for interaction. A frontend is as Web portal implemented with *Apache Cocoon* [2] and provides controlled access to B-Fabric data. Hard work is done by *(iv)* the *Workhorses* under the hood. They are used to execute specific tasks, such as indexing and searching data. B-Fabric uses *Apache Lucene* [3] for indexing and full-text searching purposes. Several frontends and workhorses can be deployed in the system in order to improve the availability and scalability. Frontends and workhorses communicate with each other via persistent message queues for asynchronous communication provided by *Apache ActiveMQ* [1]. As we will discuss in detail in Section III, arbitrary applications can be integrated into B-Fabric at run-time by using the concept of *(v) Registered Applications*. Using the registration profiles, B-Fabric dynamically extends the frontends with appropriate buttons to invoke and feed these applications with B-Fabric data. These applications may be external ones which run autonomously beyond the control of B-Fabric. Applications can also be invoked within workflows to model scientific pipelines. B-Fabric uses *OSWorkflow* [17] as workflow engine.

As shown in Figure 1, there are also several external components that interact with B-Fabric. Scientists interact with B-Fabric through frontends by using *User PCs*. These are standard computers running a Web browser which are generally used as terminals to search for and download data for analysis reasons. In addition, these PCs can also be used as workstations by deploying various data analysis and/or data visualization tools to perform the desired analysis. External systems that provide scientific data management, analysis, and/or visualization functionality correspond to *Scientific Data Marts*. B-Fabric currently uses *Rosetta Resolver* [19] and *Mascot Integra* [12] as data marts for the detailed management and analysis of transcriptomics and proteomics experiments, respectively. These data marts, on the other hand, are not suitable for some instruments data. In these cases, B-Fabric implements a custom data mart to handle such data. *External Data Stores* correspond to life sciences data available on external systems that are accessible by B-Fabric. Such data can be made available through B-Fabric by simply maintaining links to them. Ensuring consistency of the data is a responsibility of the external system as it is purely autonomous. That is, the external system must guarantee that the linked data is not deleted. Computers that are attached to an instrument that generates and holds scientific data to be imported into B-Fabric are called *Instrument PCs*. Most of the applications in the life sciences domain involves computationally intensive analysis. In order to meet this demand, B-Fabric can also execute applications on a *Computing Cluster/Grid*.

B. Metadata Schema

For the correct interpretation and reuse of experimental data, scientific annotations, i.e., metadata about the experimental data, are crucial. Finding an appropriate schema for describing this metadata is an art. On the one hand, everybody obviously agrees on the necessity of a standard schema, such as

defined by MIAME [13] or Gene Ontology [7], providing the vocabulary and ontology for the detailed description of the experimental data. On the other hand, many researchers complain that such a standard schema either does not well cover all aspects their application domain or is impractical due to various reasons, among others because standard schemas and vocabularies are not flexible enough to deal and keep pace with evolving research findings. Based on our experience with many bioinformaticians and researchers at the FGCZ and their practical experiences and difficulties with using such large standard schemas and vocabularies, we decided to apply a "minimal" metadata schema approach for B-Fabric. Figure 2 sketches the core of this schema in UML notation.

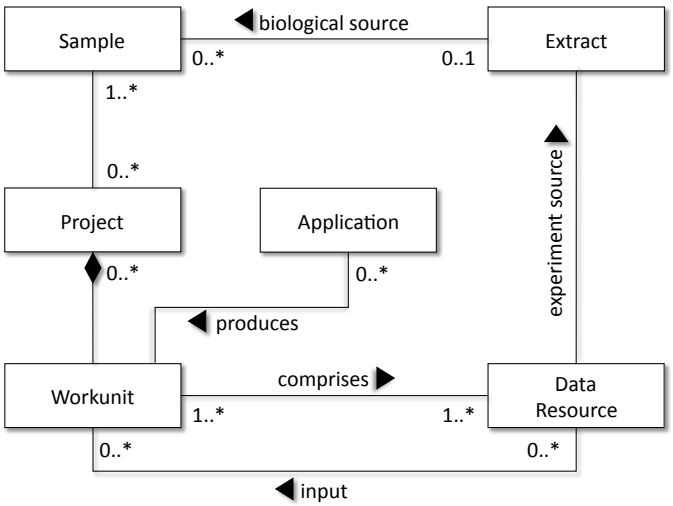

Fig. 2. Core Classes of B-Fabric's Metadata Schema

A basic concept of B-Fabric is a data resource. A *data resource* is an abstraction of a file or link to a file containing (experimental) data. That is, a data resource refers to either a file physically imported into the B-Fabric repository or a file stored outside B-Fabric but logically imported into the B-Fabric repository. Examples for data resources are raw files produced from a mass spectrometer or cel files generated from an array scanner.

Each data resource is connected to an *extract* representing the biological input into the experiment or measurement that produced the data resource. Similar to Gene Ontology, we distinguish between samples and extracts describing the biological sources at different levels. The *sample* contains general information about the biological source while the extract represents an extraction of that source which actually is used for the experiment or measurement. There might be several extracts of one sample. These extracts might be the result of different extraction procedures. To ease the finding and reuse of sample and extract information, each sample and indirectly each extract are associated with a *project*. This information helps to significantly reduce the set of values in drop-down menus, for instance, to associate a data resource with an extract.

As a result of extensive discussions between the bioinformaticians and researchers at FGCZ about the question what primary (and secondary) data should be stored and especially what this data actually represents, the generic concept of a workunit was found in B-Fabric. A *workunit* is an abstraction that can be used to represent the result of an experiment, a measurement, an analysis, a search etc. In principle, a workunit is a container referencing to data resources that logically form a unit. Some of these data resources are marked as input resources meaning that they were the inputs of the processing step that created the remaining data resources. The scientist decides on his own what a workunit should represent. Technically, it is just an object with some attributes and references to data resources.

III. Application Coupling

The concept of an application in the context of B-Fabric is to provide transparent access to all kind of functionality within the system. Data import, browsing, filtering, accessing, analysis and all kinds of processes are handled by this construct. In principle, an *application* provides a container for job execution. It consists of a set of attributes and parameters that describe how and where to access its encapsulated functionality and the data to process. By registering an application, new functionality is attached to B-Fabric and a "run" button is placed for the user on the corresponding pages of the web interface. Furthermore, applications are dynamically reconfigurable, i.e., their registration and thus their behavior can be changed at run-time. This feature allows to connect external applications to B-Fabric on-the-fly.

A. Everything is an Application

To keep the whole concept simple and to make it easy for users to understand, every processing functionality in B-Fabric is an application, whether it is a data importer that transfers data from an instrument, a data analysis tool, or a simple data transformation step. This strategy allows to bind B-Fabric internal Java methods to, e.g., binary executables, external *R* [20] applications, bash, perl, python, or ruby scripts. Users do not need to have full knowledge about how a certain functionality is implemented, they just can use it on their data and are guided through the whole process.

B. Implementation

In order to run an executable on a system, it is necessary to have a basic connector. This connector is called *worker* and is implemented in B-Fabric. As the available applications and processing pipelines are diverse (Windows vs. Unix systems, Java or Perl executables, etc.), it is a reasonable solution to add this basic connector on the B-Fabric side instead of adding a common interface to every application and processing pipeline that should be connected. Such a worker is a *Spring* [24] configured component that allows execution of a script or method on a dedicated system. For instance, the worker that connects to the Sun Grid Engine consists of about a hundred lines of Java code, using the *Distributed*

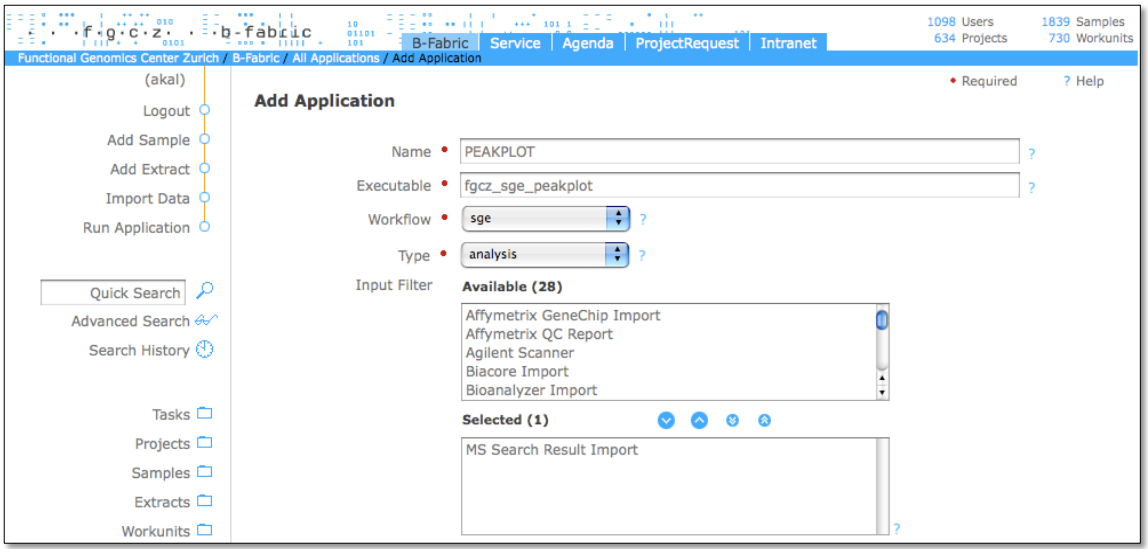

Fig. 3. B-Fabric Web Form for Application Registration

Resource Management Application API (DRMAA) [6]. The workers themselves are embedded into a workflow that handles execution and error reporting. Once such a workflow and worker exists, it is possible to register multiple applications with diverse functionality within B-Fabric without compiling or even restarting the system.

C. Application Provision

In principle, an application could be registered in B-Fabric by any user. This would however exhibit enormous risks for malicious usage of the system. Therefore, the FGCZ applies the policy that applications are coupled with B-Fabric by *power users* only. For application registration, these users own special administrator rights that let them use the corresponding registration pages of the web interface.

To provide and launch an external application the following three steps have to be done:

1) *Create Executable*
2) *Register Application*
3) *Parameter Submission*

The next paragraphs describe these steps in more detail.

1) Create Executable: First, the power user needs to make the external application available on a dedicated system, e.g. a program on the Sun Grid Engine, a method on the Rserve system, etc. In order to cooperate with B-Fabric, a small interface script has to be implemented for this application. It mainly covers how the parameters are passed on to the application and how to handle return values. As this interface connects the external application with the B-Fabric internal worker, it is worker-specific. To run an external application on a Sun Grid Engine, for example, the following parameters have to be defined:

- `<workunitId>` sets the id of the resulting workunit.
- `<inputHost>` and `<inputPath>` specify the host and location where the input files are located.
- `<outputHost>` and `<outputPath>` define the host and location where the resulting file(s) of the application should be stored.

The application has to meet the following conventions:

- It must run in a container like system environment to avoid any interaction with other external applications, e.g., writing to the same file in the scratch space. This can be solved, e.g., by using the process id in the scratch space location path.
- Moreover, it has to ensure that the copying of the input and the output resources to and from a B-Fabric storage works. For this reason B-Fabric mainly uses secure shell [16] based copy mechanisms. A solution for this would be to copy also the static linked application to the compute host. Note that it is not always easy to share huge mount-points across a "wide area" of grid infrastructures. Known hurdles are routing problems, firewalls, or network performance.
- For debugging it has to provide error handling and reporting.
- Finally, it has to provide a cleaning mechanism to avoid disk overload on the compute nodes. That is, the executable has to delete the previously copied files again.

The interface itself contains the requirement to report back a failure as soon as the external application has a condition that does not guarantee proper processing of the data. It is important for B-Fabric to know when to suspend the workflow and report an error to the user. It is recommended that the developer would test the application that has been found in this state outside B-Fabric and adapt it until it runs stable.

2) Register Application: Next, a new application object has to be created in B-Fabric to encapsulate the external functionality. This is done via a web based application registration form where the power user can choose between predefined object attributes (see Figure 3). The most important attributes

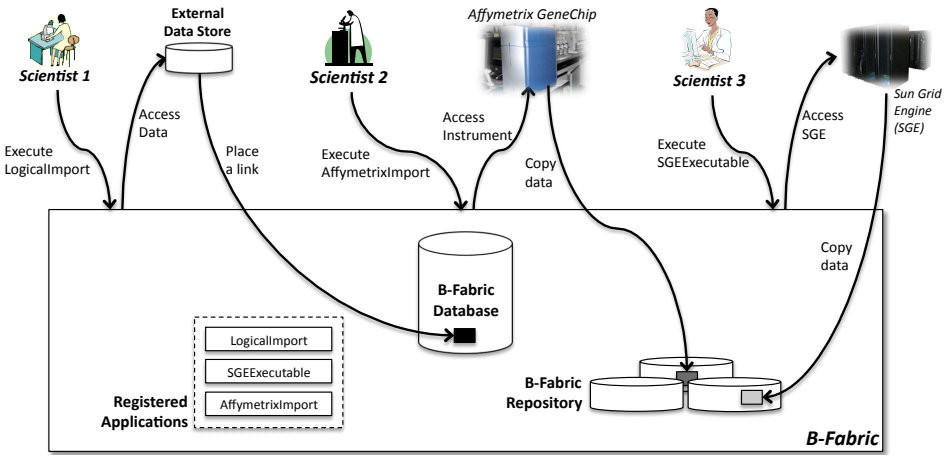

Fig. 4. User Interaction with B-Fabric Applications

that can be set to define the purpose of the application and make it work properly are listed below:

- *Name* entitles the name of the button that provides access to the functionality in the web interface
- *Executable* identifies the script, program, or method that will be called from the selected workflow.
- *Workflow* bounds each application to a certain workflow. The workflow defines how to connect to a certain job execution environment. To execute jobs, e. g., on the *Sun Grid Engine*, there exists an "sge" workflow or to run methods on an *Rserve* system [21], there is an "rserver" workflow.
- *Type* specifies whether the application provides import or analysis functionality.
- *Input Filter* is a set of applications that serves as input to this application. Data produced by applications that match to this filter are candidates for this new application. This feature allows to limit workflows on data sets that can be processed properly by the included applications.
- *Description* describes the functionality to the user and explain its benefit. This serves as a quick help.

3) Parameter Submission: Users are now able to process their data with the external application using standard parameters. If they need to define additional parameters or settings, then it is necessary to create a parameter submission form in B-Fabric. Often, certain parameters depend on other selections, e.g., the set of data resources is restricted depending on the selected projects and workunits. This makes it difficult to cover all possible selections into one generic application registration form. The straight-forward approach chosen in B-Fabric was to write an additional web form that can easily be latched onto the application execution process. This does need more programming skills, but provides a better user guidance.

Following the three steps described above allows an easy extension of the B-Fabric system. Applications which are already in use for a long time can be integrated quickly. Whereas the fast extension brings a lot of benefits for the users, it requires much discipline from the scientists side. The coupling of an application has to be tested very thoroughly to ensure that it works properly within the B-Fabric system. Especially users without proper knowledge about the applications tend to use them on incompatible or incomplete data sets. With proper application configuration and testing this handicap can be controlled. Another major benefit is the fact that all parameters and settings are stored within B-Fabric. This allows to re-run and reproduce analysis in order to verify or improve the results. Using the DRMAA interface, B-Fabric can handle massive compute jobs on a wide range of grid infrastructures.

D. Invoking Application with Data from B-Fabric

As soon as an application is registered with B-Fabric, it is accessible by the users. By default, B-Fabric automatically creates buttons on all web screens where valid data sets are available for the invocation of the application. The knowledge about which applications can be executed on what data is defined within the application itself. Each data resource in the B-Fabric system is associated with the application it was generated with. This might be an *import* application, if the data was loaded from the outside into B-Fabric or an *analysis* application, if the data was processed within the system itself.

As soon as the user clicks on the application button, B-Fabric creates a result workunit. This workunit will later hold the result resources as soon as the processing finishes successfully. For the moment it just references the input resources. This is the data set the user selected. Then, the associated workflow starts. It takes the result workunit and executes the external program with the basic parameters like the location of input and output resources. The benefit of the limited number of parameters is that it simplifies the interface. If the external application needs to have more information to process the resources, it still can query the B-Fabric database.

When the external application is executed by the workflow, B-Fabric awaits its response. If the external application fails, B-Fabric will suspend the workflow, mark the resources as "failed" and report the error back to the user. If the external application terminates successfully, the workflow will be fin-

ished and the result workunit marked as “ready”. This is the moment where the user can download and view the result.

E. User Interaction with Registered Applications

Figure 4 shows the interaction of some external components in action by exemplifying three most frequently occurring application scenarios. The arrows depict the sequential flow of interaction between a specific scientist and B-Fabric. The first scenario illustrates the case where some external data is made available via B-Fabric. To do this, *Scientist 1* executes the *LogicalImport* application. The rest is assured by this application which creates a link in B-Fabric to the data at the external data store (*logical import*). The second scenario shows how data imports are done from instruments to B-Fabric. *Scientist 2* wants to import his data generated by the *Affymetrix GeneChip* instrument. Using B-Fabric this scientist invokes a previously registered application. This application accesses the instrument and copies the data into the B-Fabric repository (*physical import*), where it will be under total control of B-Fabric. Note that when coupling an instrument to B-Fabric a corresponding import application has to be registered, too. The last scenario is an example where the user has larger computing needs. *Scientist 3* asks B-Fabric to execute his job in the Grid. To meet this demand in our case, B-Fabric accesses the *Sun Grid Engine* running on a compute cluster outside of B-Fabric. The job is executed by the cluster and the results are sent back to B-Fabric.

F. A Sample of a Pipeline of Applications

The goal of proteomics research in general is to identify and quantify proteins on a defined biochemical state. This field heavily relies on computer software and computational power. The proteome informatics workflows at the FGCZ have continuously been growing combining a wide variety of open source and commercial code at different levels of “software quality”. The workflows are serving more than 200 scientists from University of Zurich and ETH Zurich, covering a data size of more than 20 Tera Bytes, and using a cluster infrastructure with more than 300 compute nodes.

Figure 5 depicts a part of a proteome informatics workflow scenario using the introduced B-Fabric system at the FGCZ. On the left-hand side of figure, the boxes indicate the external applications while the ovals represent the data resources. On the right-hand side, the results of the corresponding applications are illustrated. Since the method of choice in proteomics is mass spectrometry, our workflows start with huge amounts of peaklists (see upper right of the figure). The first application running on these mass spectrometric measurements is a filtering method for noise reduction and peak picking [11]. This software is running in a Microsoft Windows environment because it is using Thermo Finningan instrument software libraries that are only available for this platform. The filtering is followed by an identification process which assigns in-silico digest protein sequences to the filtered peaklists using peptide fragmentation heuristics. This process cannot be triggered by B-Fabric. However, we can easily map the result data back into

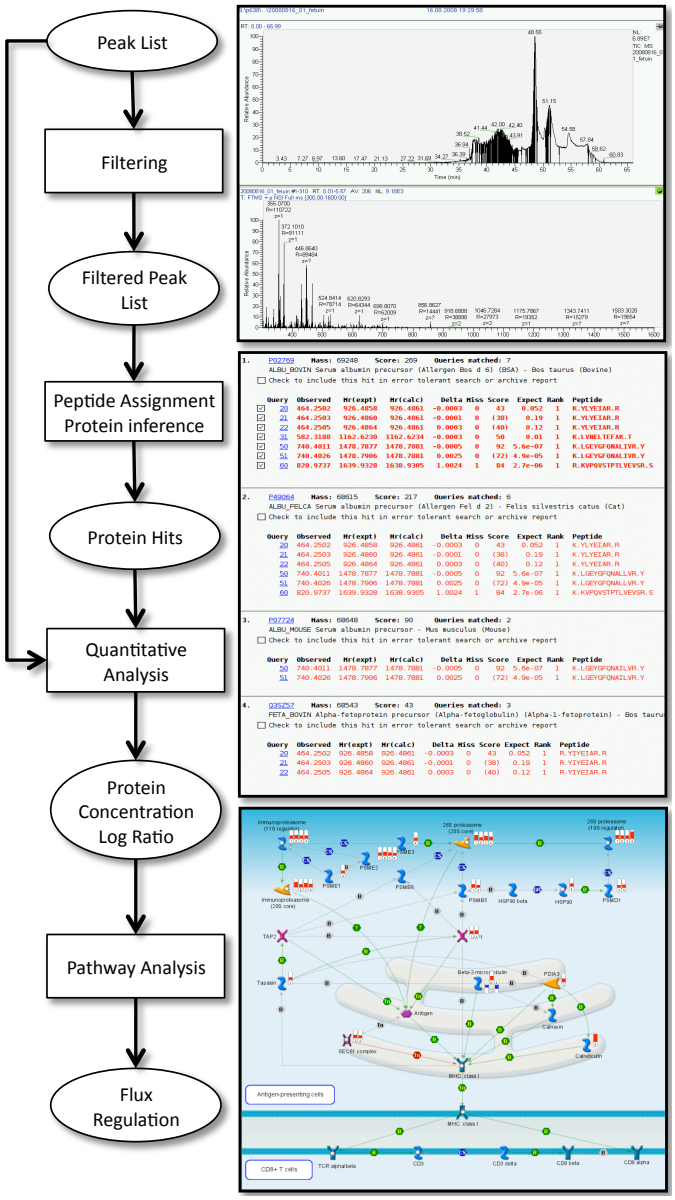

Fig. 5. Typical Proteomics Analysis Workflow based on a Mass Spectrometry Measurement

B-Fabric by using an external feeding mechanism introduced in Section IV. Quantification is becoming more and more a hot topic of interest in the proteomics research community. There exists a battery of quantification tools, e.g. [9], [22], [14], [8]. This is the reason why the users benefit most from our easy and quick application coupling approach. Finally, the quantitative information can be mapped on a biological pathway as network to indicate regulatory interactions between the proteins (bottom right).

IV. Feeding B-Fabric with External Data

In contrast to the previous section, where we have seen how loosely coupled (external) applications are fed with B-Fabric data, we now present a mechanism to feed B-Fabric with data by such applications. This important feature is motivated by

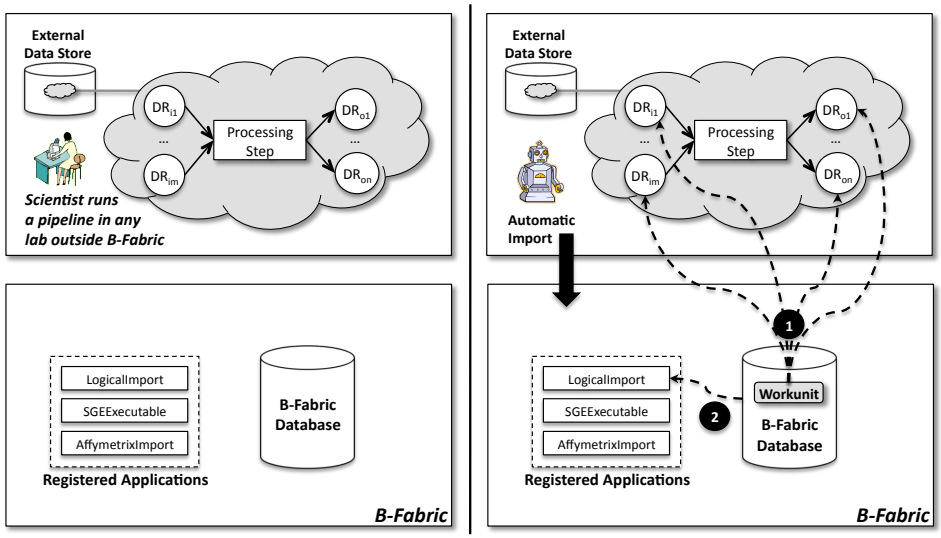

Fig. 6. Feeding B-Fabric with External Data through Robots Triggering the Direct Import Function

several scenarios that we can observe regularly at the FGCZ. For instance, we are sometimes confronted with the situation where new external data stores should be connected to B-Fabric. This is for example the case when a new partner joins the B-Fabric community and brings along existing data stores which should be integrated logically into the B-Fabric repository. More often, it happens that external applications that are not coupled yet with B-Fabric produce results which should be taken over into the B-Fabric database. For any of these situations, B-Fabric provides a generic import function at database level which allows to feed B-Fabric with external data. Figure 6 sketches how this data feeding works in B-Fabric.

A scientist runs a research pipeline in any lab outside of B-Fabric control sphere. In the figure, the pipeline is represented by the cloud in which the input data resources DR_i are used in a processing step to generate the output data resources DR_o. These data resources can be made accessible via B-Fabric using a generic import function that directly operates on the B-Fabric database. For that, there must be an external application that invokes the import function with the appropriate input parameters. The import function itself takes a set of data resources with corresponding input data resources and creates appropriate entries in the B-Fabric database. In detail, this function checks whether the data resources (identified by their location paths) were already imported. If not, it creates a workunit with links i) to the corresponding data resources on the external data store as well as ii) to the corresponding application that generated these data resources. In this way, the results of entire research pipelines can easily be represented in B-Fabric. In particular, the origin of each data resource becomes trackable. By maintaining further metadata describing the derivation history of the data resources starting from their original sources, B-Fabric supports data provenance [23].

In our current B-Fabric deployment at the FGCZ, we have a number of robots periodically checking the external data stores whether new data resources were produced. As soon as these robots detect new data resources, they import them into B-Fabric by invoking the function described above.

V. Conclusions

In this paper, we presented an approach to integrate any kind of application into the B-Fabric life science data management system. The presented approach allows a decoupling of the software development cycles of B-Fabric core system and bioinformatics applications which yield an accelerated output of new features for the users of the overall system. New applications can be plugged-in without learning the core system technology. The users can use the core system's web interface for collecting and visualizing their data using the B-Fabric annotations that are already in the system. Such typical bioinformatics applications usually need the help of an bioinformatician. On the other side, the developers of the applications do not need to take care for GUI development and data handling anymore. In addition, the data feed feature allows bioinformaticians to easily extend the B-Fabric data repository by external (often legacy) data resources. Thereby, data can be accessible very quickly and in a transparent way via a web portal. In summary, a flexible way of extending the functionality as well as the data collection of the data management system can significantly elevate the productivity at a life sciences research center like the FGCZ since the frequent and dynamically changing feature requests of scientists and bioinformaticians can be realized immediately by themselves instead of being scheduled in a long feature change list of a usually small team of hardcore informatics who is taking care of development of the core system and the overall IT infrastructure.

References

[1] Apache ActiveMQ. http://activemq.apache.org/
[2] Apache Cocoon. http://cocoon.apache.org/
[3] Apache Lucene. http://lucene.apache.org/

[4] Apache Object Relational Bridge - OJB. http://db.apache.org/ojb/
[5] B-Fabric. http://www.bfabric.org/
[6] Distributed Resource Management Application API. http://www.drmaa.org/
[7] Gene Ontology. http://www.geneontology.org/
[8] Grossmann J., Roschitzki B., Panse C., Barkow-Oesterreicher S., Schlapbach R.: Top3LFQ: Top Three Label-Free Quantification in Proteomics. 2008. Submitted.
[9] Keller A., Eng J., Zhang N., Li X. J., Aebersold R.: A uniform proteomics MS/MS analysis platform utilizing open XML file formats. Mol Syst Biol. 2005(1):2005.0017. Epub August 2005.
[10] LIMS and the art of MS proteomics. Anal Chem. 2008 Jul 1;80(13):4801-6.
[11] Mascot Distiller software. Matrix Science, London, UK. http://www.matrixscience.com/distiller.html.
[12] Mascot Integra - Data Management for Proteomics. http://www.matrixscience.com/integra.html
[13] Minimum Information About a Microarray Experiment - MIAME. http://www.mged.org/Workgroups/MIAME/miame.html
[14] Müller L. N., Rinner O., Schmidt A., Letarte S., Bodenmiller B., Brusniak M. Y., Vitek O., Aebersold R., Müller M.: SuperHirn – a novel tool for high resolution LC-MS-based peptide/protein profiling. Proteomics. 7(19):3470-80, October 2007.
[15] Oltvai Z. N., Barabasi, A.-L.: SYSTEMS BIOLOGY: Life's Complexity Pyramid. Science 298 (5594):763–764, 25 October 2002. [DOI: 10.1126/science.1078563]
[16] OpenSSH. http://www.openssh.org/
[17] OpenSymphony - OSWorkflow. http://www.opensymphony.com/osworkflow/
[18] PostgreSQL. http://www.postgresql.org/
[19] Rosetta Resolver System. http://www.rosettabio.com/products/resolver/
[20] R-Project. http://www.r-project.org/
[21] Rserve. http://www.rforge.net/Rserve/
[22] Shilov I. V., Seymour S. L., Patel A. A., Loboda A., Tang W. H., Keating S. P. et al.: The Paragon algorithm: a next generation search engine that uses sequence temperature values and feature probabilities to identify peptides from tandem mass spectra, Mol Cell Proteomics 6(9):1638–1655, 2007.
[23] Simmhan Y. L., Plale B., Gannon D.: A Survey of Data Provenance in e-Science. In: ACM SIGMOD Record 34(3):31–6, September 2005.
[24] Spring Framework. http://www.springsource.org/
[25] Sun Grid Engine. http://gridengine.sunsource.net/
[26] Türker, C., Stolte, E., Joho, D., Schlapbach, R.: B-Fabric: A Data and Application Integration Framework for Life Sciences Research. In: Data Integration in the Life Sciences, 4th International Workshop, DILS 2007, Philadelphia, PA, USA, June 27-29, 2007, LNCS 4544, pp. 37–47. Springer-Verlag, 2007.