LEVERAGING USER KNOWLEDGE
Design Principles for an Intuitive User Interface for Building Workflows

D. Kotowski, G. Henriques, M. G. Gillespie, H. Hlomani and D. A. Stacey
School of Computer Science, University of Guelph, 50 Stone Road East, Guelph, Canada

Keywords: Ontology, Ontology driven composition systems, Composition systems, User interface, Leveraging knowledge, Mind mapping, Workflow composition, Knowledge engineering, Human computer interface issues.

Abstract: Compositional systems offer a unique opportunity to users who have domain expertise but lack the necessary skills to develop software solutions in their own domain. A subset of these systems are ontology driven compositional systems (ODCS). ODCS use ontological knowledge to help facilitate composition between individual compositional units. Since an ODCS is a technologically complex system where a majority of the emphasis is placed on the inner workings of the system, often the user interface is an afterthought. This paper focuses on the human issues related to developing a workflow management application by investigating the design principles behind an ODCS interface prototype.

1 INTRODUCTION

Compositional systems offer a unique opportunity to users who have domain expertise but lack the necessary skills to implement software solutions in their own domain. The ability to compose a system from pre-existing algorithms allows for quick creation of software solutions. However, the process of software composition must be easy and approachable for the user.

A subset of compositional systems is ontology driven compositional systems (ODCS). Ontology driven compositional systems perform system composition using ontologies. The benefit of ODCS is the ability to describe software development and domain specific knowledge within an ontological representation. For example, system knowledge would be elements that pertain to the actual composition and the system on which the composition will be executed. Domain knowledge would represent information important to a user's specific domain such as aggregate data from multiple business paradigms.

An ODCS is a technologically complex system where a majority of the emphasis is placed on the inner workings of the system.Often the user interface is an afterthought. The interface needs to facilitate the interaction between the ODCS and the goal oriented mind of the user. The user does not care about the inner workings of an ODCS however they do care about having a metaphor for mapping the process that will accomplished their task. We investigate mind mapping as a way in which users can describe a workflow process of compositional units.

Current compositional systems such as Galaxy, Triana and Yahoo Pipes focus on workflow composition. They embrace the notion of designing a work flow, however are not ontologically driven. These three systems will be assessed against our usability criteria.

Overall this paper focuses on the human issues related to developing a workflow management application to mimic the task of describing a workflow. More specifically, we will describe the design process for creating the interface to our prototype ontology driven compositional system. This ODCS has been previously described in (Hlomani and Stacey, 2009) and (Gillespie et al., 2011). Systems can be viewed from many perspectives and one of the most important viewpoints is that of the user. Regardless of how the system actually works, the system metaphor that is embraced by the user (whether it is the metaphor consciously designed into the system or an implicit metaphor generated by the user's experience) is instrumental in determining if the system is successful. A successful metaphor will encourage the user to select the most appropriate system over alternatives and will allow the user to successfully complete their tasks.

In this paper we account for several factors such as usability heuristics (section 2.2), mind mapping

Kotowski D., Henriques G., G. Gillespie M., Hlomani H. and A. Stacey D..
LEVERAGING USER KNOWLEDGE - Design Principles for an Intuitive User Interface for Building Workflows.
DOI: 10.5220/0003632801760182
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2011), pages 176-182
ISBN: 978-989-8425-80-5

(section 2.3), and examination of interfaces of existing systems (section 2.4). An interface prototype was constructed using various web technologies. The basic layout of the program consists of the use of the "WireIt!" library, which is connected to an ODCS via a web service architecture. The paper will conclude with a discussion about our current prototype.

2 BACKGROUND

2.1 Ontology Driven Compositional Systems

Often there has been a disconnect between the ability to program a solution and the expertise necessary to understand the key components of a complex problem such as population modelling. The task of designing software is often out of reach of domain experts. An **Ontology Driven Compositional System** (ODCS) uses existing computational knowledge in the form of compositional units to aid a domain expert in the construction of a software systems solution (Gillespie et al., 2011). The system uses ontologies to share pertinent information between components to allow for seamless integration (Hlomani and Stacey, 2009).

An ODCS defines a workflow by combining **Compositional Units** (CU), which are discreet units of composition. A CU may be an algorithm, package or service that takes in a given input and produces a set of calculated outputs (Gillespie et al., 2011). The workflow describes the flow of data between each compositional unit as well as maintains the order of execution. The system aids the user by automatically inserting nodes that convert data formats to conform to expected input of a subsequent CU.

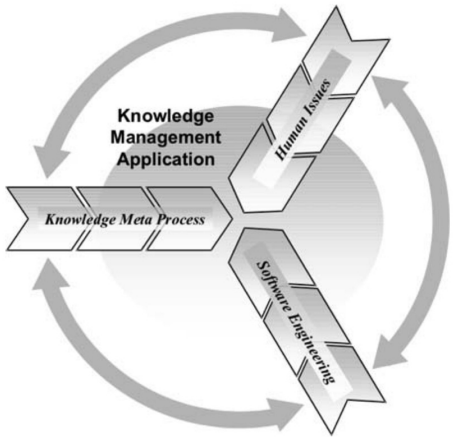

Figure 1: Processes involved in designing a knowledge management system from (Sure et al., 2009).

Since the main function of an ODCS is to compose software, the user/expert can focus on crafting the solution that best addresses the nuances of their problem domain. In essence the system leverages the knowledge of the expert to design domain specific solutions. It is important to note that many ODCS systems have been designed with a bottom up approach where the knowledge base and the underlying architecture were designed first. Until now the interfaces for these systems were minimal at best. (Sure et al., 2009) state that there are several human issues when designing a knowledge management system. Figure 1 describes the relevant processes in designing a system such as an ODCS. This paper will focus on the human issues within this process by designing an interface that helps the user leverage their knowledge within the ODCS system and that is both easy to understand and use.

2.2 Usability Heuristics

Jakob Nielsen introduced ten usability heuristics for evaluating systems and applications. One of these heuristics, flexibility and efficiency of use, states that a system should cater to experienced and inexperienced users (Nielsen, 1994). Many times, a system is composed of various types of users ranging from novices to experts.

The system aims to accommodate novice users at composing workflows of compositional units without the need of expert software knowledge. Not all users accessing the application will have the knowledge necessary to build components used within the application. For example, it is not necessary for a user to have the knowledge required to build a compositional unit which represents a statistical model. Instead, they will be able to utilize the representation of that model without having to deal with having the computational or mathematical background. At the same time, the system should provide accelerators in order to speed up interaction of expert users.

Leveraging Knowledge describes how the transfer of knowledge between two people is bi-directional and that "knowledge grows when used and depreciates when unused" (Firm et al., 2000). An example of how this concept can be used within a web application is outlined in (Gillespie et al., 2011). Here the author describes the interaction between two user types of a system. One such user can have the knowledge required to execute semantic requirements of compositional units, while another user type contains knowledge required to interconnect multiple compositional units (Gillespie et al., 2011). This exchange of knowledge is required in order to compose a system. This

concept is prevalent throughout the design of the interface.

2.3 Mind Mapping of Workflow

Mind Mapping has been used in various fields to help categorize ideas and give a visual flow between concepts (Brinkmann, 2003). The process of mind mapping allows both sides of the brain to work together to increases productivity and creativity. Mind maps are great tools for organizing information. "The hierarchical structure of a mind map conforms to the general assumption that the cognitive representation of knowledge is hierarchically structured" (Brinkmann, 2003) . This lends itself well to the notion of a compositional system. The user will lay out and connect the compositional units necessary to visualize their solution.

Mind maps excel in connecting new information with given knowledge (Brinkmann, 2003). This quality is incredibly useful when trying to expand a workflow in an ODCS. Often new algorithms or computational units will become available, and the user may choose to add them. By having a visual workflow it will be easy for the user to identify the location of where the new CU should be placed.

The mind mapping process is a logical way to represent a workflow as it gives coherence and a good visual representation. It is evident with some of the design choices in the interface presented that mind mapping was a key inspiration allowing the interface to truly leverage the knowledge of the user.

2.4 Related Existing Systems

With the acknowledgment of the different user types and their skill levels, many research initiatives have been started with the goal of meeting some of the heuristics discussed in section 2.2. In this section we provide an overview of some of these initiatives.

Triana is a problem solving environment (PSE) that can be used for composing, compiling and running applications (Majithia et al., 2004). Like many other compositional systems, the goal of Triana is to make use of several tools (data analysis tools, algorithms, control structures) to solve a problem (Taylor et al., 2007). Triana defines interfaces to a variety of execution environments through its Grid Application ToolKit (GAT) and Grid Application Protocol (GAP) allowing the execution of both task-based workflows and service-based workflows. With Triana, a user graphically composes a workflow by discovering, composing, invoking and publishing composite services in a seamless manner.

Galaxy is a tool and data integration framework (Team, 2010). It is an open-source application and allows for the installation of individual instances. The target users in Galaxy are software developers and biologists. The tools found within the application can be applied to datasets to perform calculations. New tools can be added to the interface. In order to do so, a developer must add a configuration file that contains information about how the tool is run (Team, 2010). One form of current use is as a web-based genome analysis tool (Goecks et al., 2010). The genome analysis is done in the workspace area which allows for the application of computational tools to perform the data analysis (Goecks et al., 2010). It is important to note that this application targets a specific field, biology. Furthermore, in order to keep the front-end user satisfied, it is required to maintain a person with some programming experience in case new tools are needed. Another aspect of Galaxy worth noting is that although it is a web-based application, installation only works through Linux and Mac OSX, with no current support for Windows (Team, 2010). These points play a role in increasing the cost and decreasing the adaptability of the application by other users or fields of interest.

Yahoo Pipes is a composition tool that creates data mashups; it is used primarily as a simple way to run web projects, or publish web services (Pruett, 2009). Yahoo Pipes has a set of modules that have different types of containers that can be used when building the pipe. It contains many of the concepts that are useful for CU workflow composition, however the set modules that it contains would have to be configured.

The power of ontologies is that they have the ability to automatically infer semantic relationships between CUs within the ODCS. For example, by describing the input/output parameters for each CU, the system would have the ability to infer which other CUs have the same I/O requirements. It would also have the capability of determining which CU's were created by the same software developer. In turn, this reduces the amount of work on the software development side thus reducing any dependence the front-end user might have when making modifications, such as adding new CU's, to the system.

The discussed systems are efficient in their areas and have a lot of power, but all require users with experience and background knowledge in programming in order to make modifications to fit their needs. When considering the perspective of the end user, some of the described systems are not very adaptable to change. By using an ontology-driven system, the user is able to intuitively run the system, and perform their necessary actions without being required to at-

tain the background knowledge some systems require. It also has the capability of adapting various types of fields through a given set of CU's. Using a simplistic interface, the user should be able to construct workflows of CU modules without concerning themselves with the background of the system (such as creating the CU's, defining the semantics, running the workflows, etc.).

3 ODCS INTERFACE PROTOTYPE

3.1 Goals and Principles of the Prototype

When designing the interface for our ODCS prototype system, there were several possible approaches that could have been taken. As software developers, an obvious solution would be to design a plug-in for a software IDE (Integrated Development Environment) such as Eclipse. This would add several barriers of entry upon the user. Firstly, IDE's are often resource intensive and this would force the user to invest into a substantial work station to run the software. Second, there is the learning curve involved with a complex IDE. For a developer who is familiar with IDE's it is a relatively simple learning task, however, to a non-developer it can be a long, difficult process. With such a learning curve, acceptance of an OCDS may be limited and only available to those with a strong software development background. Also, one of the main advantages of an OCDS is that the user need not be a software developer to develop software solutions, thus the IDE solution is not preferable.

Thus we centered our focus on a user that will not be experienced with software development or be particularly computer savvy. Since we assume that the user will have little to no background in software development we decided to take inspiration from a method that most user would be familiar with: mind mapping. As explained in section 2.3 mind mapping offers tools to the user to effectively create workflows. Since mind mapping is used in various fields it is very likely that the end user will have been exposed to mind maps in one manner or another. This allows for the learning curve of the user to be significantly lower.

Another factor that had to be taken into consideration was that of the access to hardware that the user may have. For this we decided to implement a distributed solution. The user interface would be provided through a web browser. This alleviates the need for high powered hardware but allows the user to still accomplish computationally intensive tasks.

As with all interfaces, usability was also taken into consideration. Described in section 2.2, the key usability heuristics focused on are flexibility and efficiency of use.

3.2 System Architecture

Our ODCS system architecture was designed in three layers. The bottom layer is the knowledge base where the main ODCS system resides. Here all the underlying logic and composition will occur. Also the physical instantiations of the CU's exist in this layer along with the ontological information. The layer above the knowledge base is a communication layer. Here a web service translates information received from the interface and presents it to the underlying knowledge base, and then once the knowledge base has processed the information it will return its results to the interface through the communication layer. Finally the top layer is the interface layer where the user interface will reside. This architecture is depicted in figure 2.

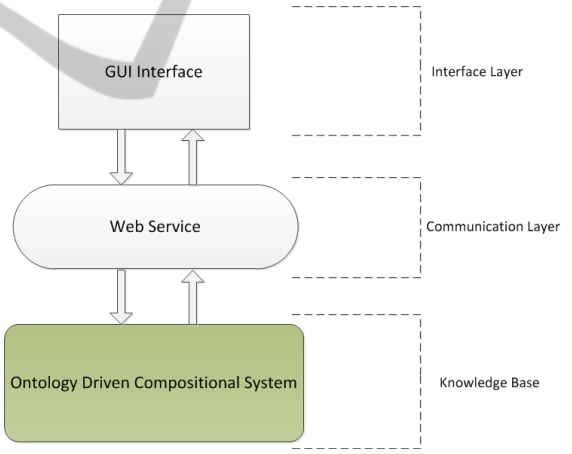

Figure 2: Conceptual Architecture of the ODCS System.

The design was inspired by modern service oriented architecture. Service Oriented Architecture (SOA) combines loosely coupled software components to generate a powerful software service(Papazoglou et al., 2007).The loose coupling of the software components allows for high portability and reuse of key software components (Papazoglou and Heuvel, 2007). In our system this means that both the interface and knowledge base are neither tied to nor dependent on one another. As long as both adhere to the web service interface they will be able to cooperate with each other. It is common among SOA's

to use web services to facilitate communications between components (Papazoglou and Heuvel, 2007).

The high adaptability of this system allows for the knowledge base to be replaced quickly and easily. This is a powerful feature as the interface is able to represent various knowledge bases describing different composition systems. All that is necessary is the conformity to the communication layer.

3.3 WireIt! for Workflows

WireIt! is an open source Javascript library which is used to create web "wireable" interfaces for data-flow applications, visual programming languages, graphical modelling, and graphical editors (Abouaf, 2010).

It has been adapted as the interface for ODCS due to the flexibility and adaptability that this library exhibits. *WireIt!* introduces many different concepts that can be used within applications such as containers and wires. There are various types of containers: image containers, input/output containers, form containers, etc.. It also has the flexibility for a developer to create their own container with set properties.

WireIt! also presents various examples of how their components are used to build a system. In particular, the wiring editor example was used as the base of the interface for the ODCS. Wirable I/O containers were used as the main representation for CU's. A panel within the wiringEditor provides a section where workflow composition can be performed. The presentation of workflows through the use of this library can be compared to the visual flow of concepts that is commonly seen with mind-mapping applications as described in section 2.3. A further breakdown of how and why *WireIt!* was adapted to the ODCS is seen in section 3.4.

3.4 GUI Components

As mentioned in the above section, the *WireIt!* library's wiringEditor was used as the base of the ODCS interface. The system uses a four panel layout; header, left, body, right. The following sections will describe the functionality and purpose of each panel.

The layout is depicted in figure 3.

3.4.1 Header

The header contains a toolbar with various functionality to assist the user in uploading and storing workflow schemas; New, Load, Save, Delete, Help. The New function creates a blank workflow. The workflow is loaded from an XML file which is passed from the backend to the interface through a web-service.

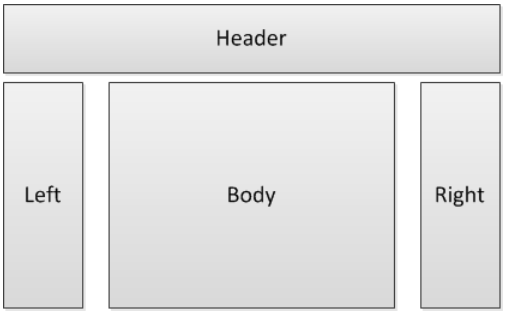

Figure 3: ODCS Interface Layout.

The file is then parsed and displayed in the workflow area. By using this information, the user does not need to concern themselves with the process of loading a workflow schema, *i.e.* they do not have to build the XML file themselves; this will already have been done when a workflow is saved. The Save function saves the workflow composition that the user has built. The composition is saved in an XML template file and passed to the backend of the application. Leveraging knowledge is seen here, as the software developers are using the expert knowledge the end user has to put together the workflow. In turn, the user is using the knowledge of the software developer in order to save the configuration of the current workflow. Delete will remove all content from the existing workflow. If the workflow has been saved, a blank XML template file will be passed to the backend of the application and this configuration will be saved. If the delete is used before saving content, the user is simply presented with a blank workflow without the need of going through the backend of the application. The Help menu option displays simple instructions for the end user on how to compose workflows within the application. The Help menu option displays simple instructions for the end user on how to compose workflows within the application.

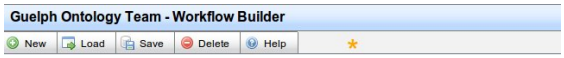

Figure 4: ODCS Interface Header Menu.

3.4.2 Left Menu

The Left panel contains a Menu holding the CU Modules collection. The modules are categorized and sorted by CU purpose and models.

The CU modules are representations of different knowledge frameworks that a user may require. For example; some modules may be statistical models, simulations, visualization software, etc. The user uses these modules to build a workflow. They are not re-

quired to create the CU themselves. The CU's are developed by software developers, and at times, the creation of these CUs requires expert knowledge from other disciplines. The user does not have to worry about going through a learning curve in order to utilize the CU's; nor do they have to worry themselves about the complexity found within the creation of these modules. Thus leveraging of knowledge proves to be a cost efficient and time saving process for the front end user. It is also important to note that the application is not bound to a single set of modules.

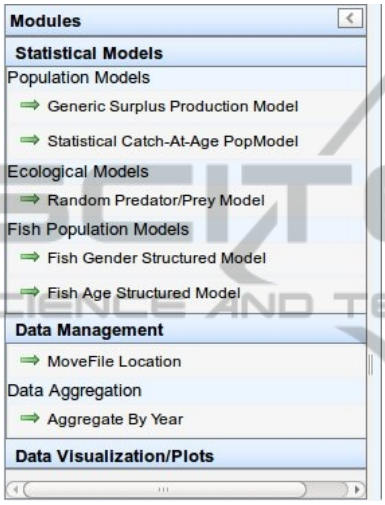

Figure 5: ODCS Interface Header Menu.

3.4.3 Body

The body of the application contains a workflow builder area. This is a section where users can construct CU workflows. The information in this area can be loaded and stored. A workflow is composed of:

- **I/O Container.** This is the representation of a compositional unit (CU). It contains a set of input and output terminals that display CU parameters. Input terminals are on the left side, output terminals are located on the right side.
- **Wires.** Wires are used to connect the I/O Containers. A wire may only connect an output terminal from one container to the input terminal of another.

The concept of mind mapping was used as a template for the workflow builder area. The CU workflow composition provides a visual representation of existing knowledge (a CU module) being used in the construction of new information (multiple CU modules connected together forming a new concept).

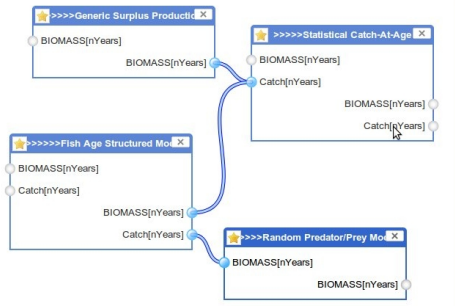

Figure 6: ODCS Interface Header Menu.

3.4.4 Right Menu

This section is split into three different categories: Info, Properties, and Workflow Map. The Info section displays information about the workflow builder application.

The Properties section displays information about the CU semantics. Each CU module contains a set of properties that make up the CU's semantic knowledge. These semantics are categorized by type: Trust, Quality, and Functionality. This information is based on the current set of CU module collections found on the left panel of the application.

The workflow map section displays a mini map of the current workflow. This mini map allows the user to navigate through the workflow builder area by clicking on a section in the map and dragging it to the desired spot.

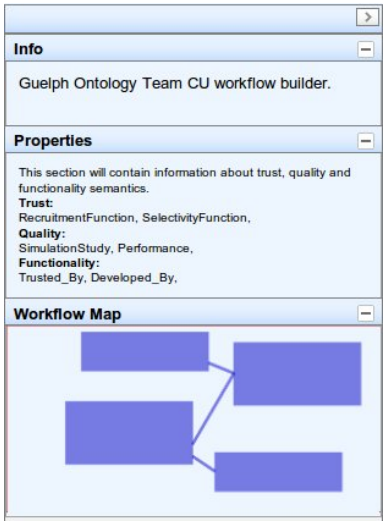

Figure 7: ODCS Interface Header Menu.

4 DISCUSSION & CONCLUSIONS

In this paper we have exposed the metaphor and design principles used in our implementation of a prototype interface for our ontology driven compositional system. The prototype implemented the notion of mind mapping as a metaphor for workflow composition. From this prototype we have identified a clear convergence of mind mapping and workflow composition. With this method it was possible to deliver an interface that leveraged the user knowledge with the knowledge contained in the system.

As examined within section 3.1, an IDE plug-in is an alternative way to approach this problem. However the cost of entry would be exorbitantly high for users that do not come from a software development background. It is important to emphasize that the purpose of an ODCS is to enable domain experts to development software who do not necessarily have software development skills. This is why we decided to focus on the metaphor between mind mapping and workflow composition. Many people have been exposed to mind mapping in some form or another and thus this lowers the barrier of entry to the average user. Users will see value in the system when it is easy to learn and understand.

Now that a working prototype has been established it is necessary to examine whether or not this method will suite users/experts from different domains. User testing must examine if the heuristics presented are being met in a wider user base.

The human Issues involved in designing a knowledge based system must be examined thoroughly and solutions must be tailored to the user base. Level of entry as well as access to hardware were driving factors within our design choices. We do not dispute there are many ways to approach this problem, however, we strictly tailored our approach to that of a domain expert and the average user. This will open the system to many users from many different domains.

ACKNOWLEDGEMENTS

We would like to acknowledge all the work of the Guelph Ontology Team as this paper would not be possible with out their collaboration and hard work. We would also like to thank Deb Stacey for her support and guidance with this work.

REFERENCES

Abouaf, E. (2010). Wireit a javascript wiring library. Online: http://neyric.github.com/wireit/.

Brinkmann, A. (2003). Graphical Knowledge Display Mind Mapping and Concept Mapping as Efficient Tools in Mathematics Education. *Mind*, pages 35–48.

Firm, T., Chain, V., and Network, V. (2000). Ten Ways to Leverage Knowledge for Creating Value. *Knowledge Creation Diffusion Utilization*.

Gillespie, M. G., Stacey, D. A., and Crawford, S. S. (2011). *Designing Ontology-Driven System Composition Knowledge and Processes to Satisfy User Expectations (in publication)*. Lecture Notes in Computer Science. Springer-Verlag.

Goecks, J., Nekrutenko, A., Taylor, J., and Galaxy Team, T. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86.

Hlomani, H. and Stacey, D. A. (2009). An ontology driven approach to software systems composition. In *International Conference of Knowledge Engineering and Ontology Development*, pages 254–260. INSTICC.

Majithia, S., Shields, M., Taylor, I., and Wang, I. (2004). Triana: a graphical Web service composition and execution toolkit. *Proceedings. IEEE International Conference on Web Services, 2004.*, pages 514–521.

Nielsen, J. (1994). Usability Engineering. *San Diego: Academic Press*, pages 115–148.

Papazoglou, M. P. and Heuvel, W.-J. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415.

Papazoglou, M. P., Traverso, P., Dustdar, S., and Leymann, F. (2007). Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45.

Pruett, M. (2009). Yahoo Pipes.

Sure, Y., Staab, S., and Studer, R. (2009). *Ontology Engineering Methodology*, pages 135–152. Springer Berlin Heidelberg, Berlin, Heidelberg.

Taylor, I., Shields, M., Wang, I., and Harrison, A. (2007). The triana workflow environment: Architecture and applications. *Workflows for eScience*, pages 320–339.

Team, G. D. (2010). Galaxy Project Page. Online: http://usegalaxy.org.