# HyperThesis: The gRNA Spell on the Curse of Bioinformatics Applications Integration

Sourav S Bhowmick
School of Computer Engineering
Nanyang Technological University
Singapore 639798
assourav@ntu.edu.sg

Vivek Vedagiri
HeliXense Pte Ltd
Science Park
Singapore 110254
vvedagiri@helixense.com

Amey Laud
HeliXense Pte Ltd
Science Park
Singapore 110254
alaud@helixense.com

## ABSTRACT

In this paper, we describe a graphical workflow management system called *HyperThesis* to address the challenges of integrating bioinformatics applications. HyperThesis is an integral component of the *Genomics Research Network Architecture* (gRNA). The gRNA was designed and developed to address the challenges of developing new bioinformatics applications. Specifically, HyperThesis makes constructing workflows (pipelines of execution of applications) in the gRNA fast and intuitive for biologists and bio-programmers alike. It provides a large repository of interconnectable, parameterized *workflow components* for processing and relating diverse biological data and software programs. It also enables us to add new workflow components as new algorithms develop in ones area of interest. HyperThesis has been fully implemented using Java.

## Categories and Subject Descriptors

J.3 [**Life and Medical Sciences**]; H.2.8 [**Database Management**]: Database Applications; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

## General Terms

Management, Human Factors, Design

## Keywords

Bioinformatics, applications integration, work flow, HyperThesis, GUI, visual components, XomatiQ, gRNA.

## 1. INTRODUCTION

The amount of biological data being generated worldwide is growing rapidly, as is the need for analyzing and managing

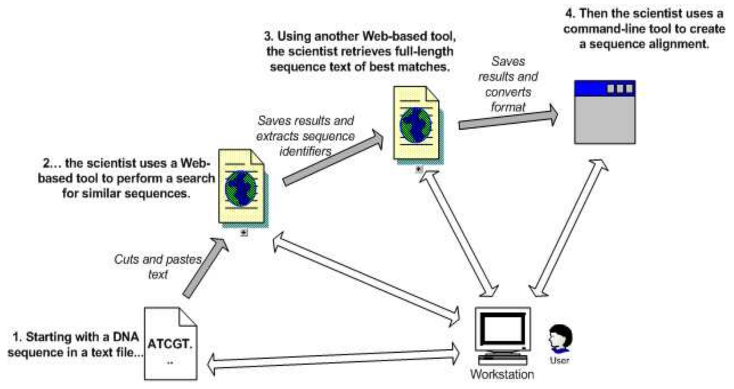
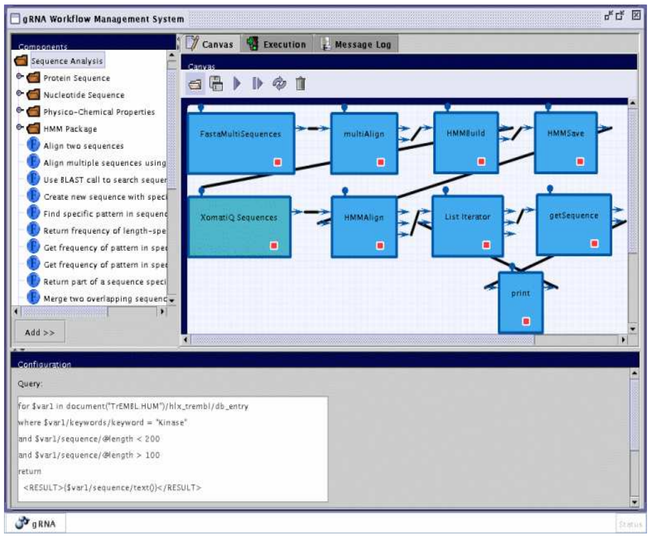

3. Using another Web-based tool, the scientist retrieves full-length sequence text of best matches.
4. Then the scientist uses a command-line tool to create a sequence alignment.
Saves results and converts format
Saves results and extracts sequence identifiers
2... the scientist uses a Web-based tool to perform a search for similar sequences.
Cuts and pastes text
1. Starting with a DNA sequence in a text file...
ATCGT.
Workstation
User

**Figure 1:** Typical sequence of steps.

this wealth of information. This data, which lies at the foundation of hypothesis development and experiment validation, is often stored in geographically distributed databases (eg. EMBL, GenBank, SwissProt etc.). These databases contain annotated genomic sequence information (EMBL, Genbank), or the results of new high-throughput techniques such as microarray experiments, or *curated* databases containing carefully scrutinized existing research, systematically compiled by domain experts [17]. It is critical for biomedical and life science researchers to correlate these databases with those containing medical records, information on disease, references to literature, information on the properties of chemicals and their molecular structure.

Most of these data is voluminous, highly evolving, highly heterogeneous, semi-structured, and not consistently represented. This heterogeneity results from our evolving understanding of complex biological systems; the numerous disparities in modeling biological systems across organisms, across tissue, in different environments and over time; and disparities across the scientific community in their understanding of these systems. Moreover, there continues to be a lack of common standards in the representation of biological data [4].

There is also a plethora of bioinformatics software tools currently available. Many of these tools are freely available on the Web. For example, the BLAST or FASTA programs can be used to look for sequence similarity, InterProScan [5] is a tool that can be used to search for motifs in protein


gRNA Workflow Management System
Components
Sequence Analysis
Protein Sequence
Nucleotide Sequence
Physico-Chemical Properties
HMM Package
Align two sequences
Align multiple sequences using
Use BLAST call to search sequer
Create new sequence with speci
Find specific pattern in sequenc
Return frequency of length-spe
Get frequency of pattern in spec
Get frequency of pattern in spec
Return part of a sequence speci
Merge two overlapping sequenc
Add >>
Canvas
Execution
Message Log
Canvas
FastaMultiSequences
multiAlign
HMMBuild
HMMSave
XomatiQ Sequences
HMMAlign
List Iterator
getSequence
print
Configuration
Query:
gRNA
Status

```
for $var1 in document("TrEMBL.HUM")/hlx_trembl/db_entry
where $var1/keywords/keyword = "Kinase"
and $var1/sequence/@length < 200
and $var1/sequence/@length > 100
return
 <RESULT>{$var1/sequence/text()}</RESULT>
```

**Figure 2:** HyperThesis GUI.

sequences, DALI [1] can be used to query protein structures and compare them to those in the Protein Data Bank (PDB), and ClustalW [20] can be used to perform sequence alignment. Most of these tools have different client interfaces (one interface for each server-side program) which are made available to all potential users. However, this implies that a user cannot customize the interface to his/her specific requirements or preferences.

While these databases as well as these tools are suitable for single pass analysis, such as comparing a new sequence against a public database or calculating the reverse complement of a sequence, what is required is the ability to use the results of some analysis as the basis for conducting further downstream analysis (eg. identifying new genes) in a manageable, efficient way. However, with the diverse range of file formats, different computer platforms, as well as representations of bioinformatics data, it has become an increasingly daunting task to work with different analysis tools. Researchers wishing to perform multiple-pass analysis of data, by feeding results of one program to another, encounter the problem of changing data from one format to another. For instance, Figure 1 [18] describes a typical sequence of steps a scientist might take to accomplish a relatively simple analysis task. To examine a DNA sequence in alignment with similar sequences from a public database, the scientist must use three different tools with three different interfaces and convert the output from each one to a format acceptable as input to the next. For many complex analysis, many other resources might be required. Consequently, scientists find it slow, cumbersome, and labor-intensive to establish connections across information resources that fuel scientific research. It has been claimed in [18] that biologists spend more than half of their time on tasks related to the integration of data from incompatible databases and software tools.

Recognizing that integrating geographically distributed heterogeneous biological data sources and different bioinformatics tools is essential, there arises the need to develop a standardized but flexible framework for integrating different types of biological data (eg. primary sequences, functional data, ESTs) and software programs. `HyperThesis` was designed and developed at HeliXense Pte Ltd, Singapore [2] to achieve this goal.

`HyperThesis` is a graphical workflow management system for bioinformatics applications that represents sophisticated functionality as intuitive *workflow components*. By interconnecting workflow components the user can create data-processing workflows which perform complex, large-scale processing automatically. The components are arranged and connected based on an experimental approach to the task to be accomplished, rather than a programmatic one. `HyperThesis` allows developers to easily design complex workflow components since it allows them a great deal of flexibility in modularizing and parameterizing code into workflow components. Note that `HyperThesis` is data flow driven.

`HyperThesis` is an integral part of the *Genomics Research Network Architecture* (gRNA) [1] [15] that was designed and developed to address the challenges for developing new bioinformatics applications. The gRNA provides a *development environment* and a *deployment framework* in which to maintain distributed warehouses, and to model, query and integrate disparate sources of data. The reader may refer to [15] for a detailed discussion on the architecture and application of the gRNA.

The rest of the paper is organized as follows. We discuss the logical structure of `HyperThesis` in Section 2. In Section 3, we present the `HyperThesis` interface and discuss how the anatomy is implemented in the gRNA. Then, in Section 4 we illustrate with an example how `HyperThesis` can be used to integrate bioinformatics applications. We discuss related work in Section 5. Finally, the last section concludes the paper.

## 2. ANATOMY OF HYPERTHESIS

In this section, we present the anatomy of `HyperThesis`. `HyperThesis` is logically composed of two main parts: a collection of *workflow components* and the *execution engine*. The workflow components are the basic units of work within `HyperThesis`. Each component is built to perform a specific operation based on the inputs and parameters supplied to it. The *execution engine* is the lifeline of `HyperThesis` in the sense that it performs the most essential task: execution of a collection of workflow components. We now elaborate on these two parts.

### 2.1 Workflow Components

The workflow components are the basic building blocks of `HyperThesis`. `HyperThesis` provides a rapidly growing large repository of interconnectable, parameterized workflow components that opens a wide range of possibilities for processing and relating diverse biological data. This repository contains sophisticated algorithms for sequence, expression and structure analysis (e.g. BLAST, Hidden Markov Models (HMM), clustering methods, Self Organizing Maps (SOM), Support Vector Machines (SVM), and many more). One can also expand the pool of workflow components in `HyperThesis` simply by adding new user-defined workflow components into this repository. Using an assortment of them, one can then weave together a bioinformatic workflow that performs a particular task as desired by the user. Each

[1]The first version was officially released on 3rd Jan, 2002.

workflow component in HyperThesis is designed to perform a specific task such as reading a sequence from a file, performing alignment on a given set of sequences, etc. Most workflow components have *inputs*, *parameters* and *outputs*. Outputs of one component can be connected to inputs of other components and so on to form complex workflows.

The workflow components follow the object oriented paradigm in the sense that all components fall into one of two basic types. They either involve the instantiation of a new object, or execution of a method of an object that has been passed in via its input. There would be components that perform both these tasks, but the fundamentals are the same i.e., to create objects and make use of the methods they provide. For example, in Figure 2 the boxes represent workflow components. The screenshot shows how a set of workflow components can be connected to perform a task (multiple alignment of sequences [16]). While each component appears to be a "black-box", it actually represents code written in Java or Python. The components communicate with each other through their inputs and outputs. During such communication the components actually pass in and out Java (or Python) objects. We will elaborate on the HyperThesis GUI in Section 3.

## 2.2 HyperThesis Execution Engine

The workhorse of HyperThesis is the *execution engine*. The job of this engine is to execute the code written in each of the workflow components in an orderly manner. All workflows are processed automatically. The execution environment is implemented as a Java application triggered by any event on the *user interface* (discussed later), such as start, stop, step, etc. The engine supports code written in Jython (a seamless interface between Java and Python).

In HyperThesis terminology, a workflow component is "ready for execution" only if all its object inputs (i.e. inputs that are derived from the output of another component) have been *satisfied*. An object input is *satisfied* if the workflow component from which it is obtained has already been executed. In that case, the engine takes care of transferring the object from the executed workflow to the object input.

The working of the engine can be described in the following steps:

1. Scan through all the workflow components to find one that is ready for execution. (At the start of execution, obviously, only a workflow component with no object inputs will qualify as "ready for execution").
2. If no such workflow component is found, then the workflow has been completely executed. Otherwise, execute the code in the workflow component.
3. Once execution of one workflow component is done, the outputs of that workflow component are collected and passed to the next set of components (to which the outputs are connected). After that the engine will find the next workflow component ready to be executed (all of its inputs are available) and execute it. Once done it will pass the outputs to the appropriate workflow components, and so on.
4. The above process is repeated until there are no more workflow components that are to be executed and the workflow execution is deemed finish.

It might seem that if two workflow components are likely to be simultaneously ready for execution then their order of execution is ambiguous. This is in fact true, but HyperThesis provides a mechanism for averting this ambiguity through the provision of *activation inputs*. These inputs determine whether the workflow component is "activated" or not. A workflow component is "activated" only when one of two conditions occurs: (1) when there is no input to the activation input and (2) when there is an input to the activation input, and the workflow component from which this input is obtained has been executed. The HyperThesis engine executes only "activated" workflow components. Therefore, activation inputs can be used to ensure that certain workflow components execute only after certain other components have been executed.

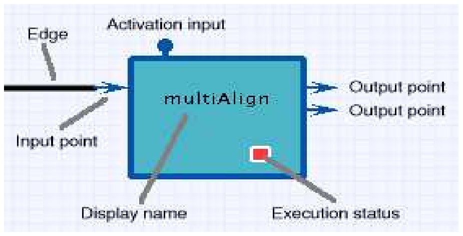


**Figure 3:** A Visual Component.

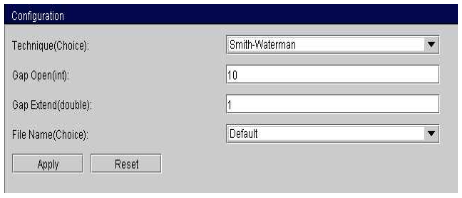
## 3. INTERFACE OF HYPERTHESIS

We now discuss how the two parts of HyperThesis discussed in the preceding section are realized in the gRNA. Specifically, we discuss the GUI of HyperThesis. Note that we do not discuss the implementation of the HyperThesis interface in great detail as the detailed description is proprietary and hence beyond the scope of the paper.

Figure 2 shows the graphical interface of HyperThesis running on the gRNA platform. The HyperThesis user interface is implemented as a Java application using Swing as the user interface toolkit. The interface consists of three parts as discussed below.

The *Working Canvas* occupies the centre of the window. This is the area where users can drag *visual components* around and construct an execution pipeline from various components. The canvas visualizes the execution flow of the program and enables intuitive understanding of the executed operation.

On the left side is the *Component Palette*. This palette contains all the available *visual components*. A visual component (hereafter, unless otherwise specified we will use the terms component and visual component interchangeably) is the implementation of a workflow component in the HyperThesis (discussed in the preceding section). Hence, visual components are the basic building blocks of any workflow in HyperThesis and are basically an abstraction of certain operation into easily managed visual representation that can be reconfigured and reconnected on the fly to execute a larger and more meaningful task. Conceptually, there are two parts to every visual component. The first part is the *configuration*. A visual component should know how to render its *configuration panel*, so that information specific to its operation can be entered by users. The second part is the definition of its execution behavior. Given the appropriate parameters and inputs, visual components should follow pre-defined steps when the operation is executed. We shall discuss visual components in detail in Section 3.1.


Configuration

Technique(Choice): Smith-Waterman

Gap Open(int): 10

Gap Extend(double): 1

File Name(Choice): Default

**Figure 4:** Example of configuration panel.

The *Configuration Panel* occupies the bottom of the screen. When a visual component is added to the working canvas, this component needs to be configured. In most cases, the configuration is specific to each component. For example, the configuration panel in Figure 2 is specific to the *XomatiQ Sequence* component in the workflow. To execute this component, one has to specify a query in the configuration panel to access relevant data from specified biological source(s) (discussed later in Section 4). Figure 4 shows another example of configuration panel.

## 3.1 Visual Components

As mentioned earlier, the visual components are the representation of the workflow components in the `HyperThesis` GUI. Each visual component added to the working canvas is represented as a blue box. We first discuss the anatomy of such a visual component and then elaborate on how they can be used to create meaningful workflows.

Figure 3 depicts the various parts of a visual component. The *display name* (*multiAlign* in Figure 3) is a good indication of the function of the component. In the above example, it is used for multiple alignment of genomic sequences [16]. The *input points* are to be connected to outputs of other components in order to facilitate the flow of objects from one component to another. In most cases a component will only be executed if all of its inputs are valid. There are some exceptions such as *Loop* and *Mux* components where the component will be executed as soon as one of its inputs is valid. The *Loop* component allows a certain workflow to be executed multiple number of times. The number of times the loop is to be performed is specified in advance and cannot be modified during runtime. The *Mux* component is used for multiplexing two inputs. The output of this component will be the first input if both inputs are available, or the active input if only one of the inputs is active. The *activation input* is a special type of input that determines whether a component is "activated" or not. As described in an earlier section, it is used to impose a specific order of execution of components. The *output points* provide the outputs for this component. *Edges* represent the data flow in the pipeline. An edge from the output of a component to the input of another component represents that the object originating from the output of the first component is carried through and used as an input parameter of the second component. The *execution status* signifies the status of the component during execution. A red box indicates that the component has not been executed yet, while the green box shows that the component has been executed.

## 3.2 Adding Existing Visual Components

`HyperThesis` provides a large repository of visual components to the user for creating bioinformatic workflows. In this section, we illustrate with an example how one can easily

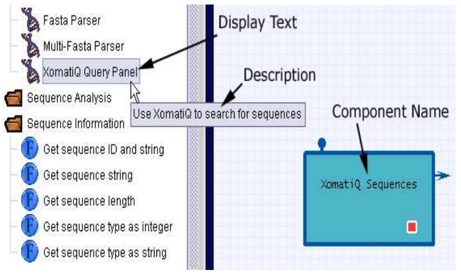


**Figure 5:** Component Palette.

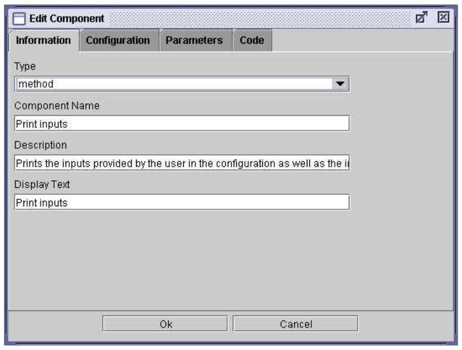
add these existing visual components to the Working Canvas and create a workflow to perform a specific task. Consider the workflow in Figure 2 which is created to perform multiple alignment of sequences (We elaborate on the semantics of the workflow in Section 4). The *XomatiQ Sequences* component is used to retrieve specific set of sequences from the TrEMBL database [7]. This component is imported to the Working Canvas by selecting the "XomatiQ Query Panel" component in the Component Palette and clicking on the "Add" button (Figure 5). Once the component is visible in the canvas, one can drag the component around and connect it to other components. When one clicks on the added component the configuration panel will automatically display a form so that information specific to the execution of the component (if any) can be entered . In the above example, the panel enables us to enter a query to retrieve relevant data (Figure 2). Similarly, other components can be selected from the Component Palette and connected to form a workflow for a specific task. `HyperThesis` supports automatic type checking. When one connects two components, the types of objects being output from one component and accepted as input by the other component are checked for compatibility. If they are not matching, an error message is displayed. Observe that the ease of connecting up components and setting their parameters does not require users such as biologists to grapple with the logic of what actually happens "behind the scenes".

Once the workflow is ready, it can be saved and executed multiple times. Every time one needs to modify the workflow, (s)he only needs to click on the component and change the configuration without recreating the entire workflow. Upon using `HyperThesis`, it becomes obvious how easy it is to translate one's ideas into a meaningful workflow. Instead of coding and debugging programs or scripts, a biologist can concentrate on the formulation of the problem and solve it quickly.

## 3.3 Creating new Visual Components

There will be instances when the components in the Component Palette are insufficient to meet ones needs or inappropriate for use. For example, one may wish to implement a new algorithm which (s)he has designed for a specific task. Consequently, one would wish to add a new component to the Component Palette. Hence, we now discuss how a new visual component can be added in the `HyperThesis` component repository.

The process of adding a new user-defined visual component requires the user to first provide *certain* information about the component. After that, the user also needs to supply information about where within the `HyperThesis` com-


Edit Component

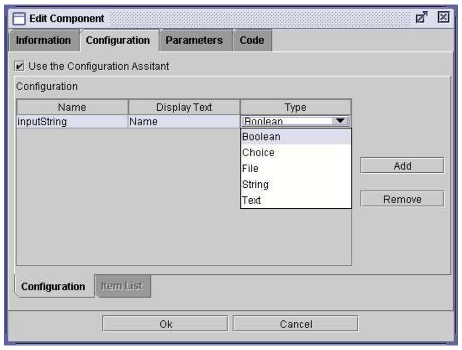
Type: method

Component Name: Print inputs

Description: Prints the inputs provided by the user in the configuration as well as the i

Display Text: Print inputs

**Figure 6:** GUI for *information* about components.


Edit Component

☑ Use the Configuration Assistant

Configuration

| Name | Display Text | Type |
|---|---|---|
| inputString | Name | Boolean |

**Figure 7:** GUI for *configuration* about components.

ponent palette the new component is to be placed. We now elaborate on how these tasks are performed in `HyperThesis`.

## *Providing Component-related Data*

In order to add a new component to the `HyperThesis` component repository , the user needs to supply the following data: *information*, *configuration*, *parameters*, and *code*. We discuss how this is achieved by using the `HyperThesis` *Component Manager*.

**Information:** This category refers to the general information about the component. In specific, the following four types of information are required:

- *Type:* This refers to the type of the component. The type of a component can be either a *constructor*or a *method*. A *constructor* is used to indicate the fact that the component is one that creates an object. On the other hand, a *method* is a component that takes in an input object and executes some methods with it. In most cases, a *method* also returns a result.
- *Component Name:* This refers to the name of the component. Whenever the component is dragged on to the `HyperThesis` canvas, this is the text that is displayed within the blue box. In Figure 5, the component name is *XomatiQ Sequences*.
- *Description:* This refers to a brief description of the component's functionality. Figure 5, depicts the description of the component *XomatiQ Sequences*.
- *Display Text:* This refers to the text that is displayed in the tree shown in the component panel of the `HyperThesis` GUI. For example, the display text of the *XomatiQ Sequences* component in Figure 5 is *XomatiQ Query Panel*.

The information specified above is entered into the *Edit Component* window of the component manager. A screen shot of this window is shown in Figure 6.

**Configuration:** For each component, there is a configuration interface associated with it. For simple user inputs such as strings, numbers, *choice*, the `HyperThesis` automatically creates the interface. For more complex inputs that are not supported directly by the `HyperThesis`, the user would need to create classes that display the required interface, gather the inputs and process it. Once these classes have been created, the user only needs to uncheck the "Use the Configuration Assistant" checkbox and enter the names of these classes in the configuration tab of the *Edit Component* window (Figure 7).

| Type | Description |
|---|---|
| Boolean | True or False |
| Choice | One out of a given list of choices |
| File | The name of a file |
| String | A string value (in one line) |
| Text | A piece of text (multiple number of lines) |

**Figure 8:** Types of input.

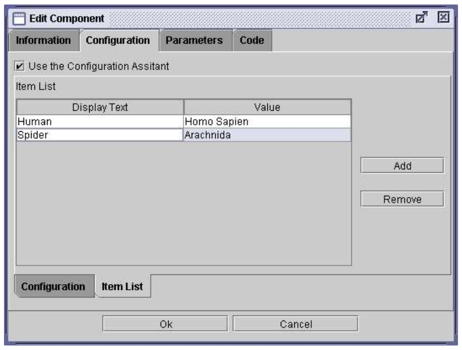
Each row within the interface corresponds to one input on the configuration panel. To define an input, the following fields are required:

- *Name:* This the variable name assigned to this particular input. This variable name can be used in the Jython code for this component.
- *Display Text:* This refers to the text in the label alongside the input, on the configuration panel.
- *Type:* This refers to the type of input that is required from the user. Figure 8 shows the different types that are currently supported in the configuration. Depending on the type chosen, the actual GUI component displayed by the gRNA on the configuration panel varies. For example, if the input type is *choice*, a drop down list is displayed whereas if the input type is *string*, a text box is displayed on the configuration panel.

Out of the different types of configuration inputs, the *choice* input is special in that it requires a list of choices to be provided when the component is being created. For this purpose, a tab labelled *Item List* is enabled whenever a configuration input of type *choice* is selected (Figure 9).

**Parameters:** Parameters define how the component interacts with other components. In other words, they are the inputs that the component receives and the outputs that it provides. For every parameter, the following fields are specified (Figure 10):

- *Name:* This the variable name assigned to this particular parameter. This variable name can be used in the Jython code for this component.
- *Display Text:* This is the text that is displayed when the user moves the mouse over the input or output arrow that corresponds to this parameter, when the component is displayed on the `HyperThesis` canvas.
- *Type:* This refers to the class to which the parameter belongs. In case the parameter is an input parameter, the *type* field should contain a list of all the classes that are compatible with this input.

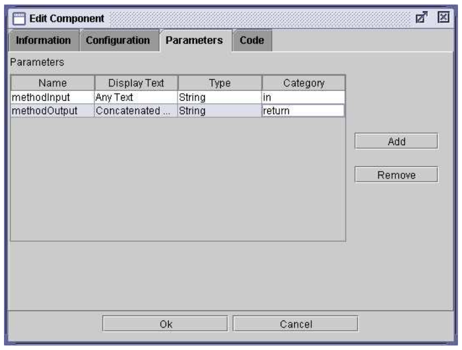


☑ Use the Configuration Assitant

Item List

| Display Text | Value |
|---|---|
| Human | Homo Sapien |
| Spider | Arachnida |

**Figure 9:** GUI for *item list*.



Parameters

| Name | Display Text | Type | Category |
|---|---|---|---|
| methodInput | Any Text | String | in |
| methodOutput | Concatenated ... | String | return |

**Figure 10:** GUI for *parameter* definition.

- *Category:* The `HyperThesis` Component Manager allow two types (or categories) of parameters; **in** and **return** that corresponds to input and output parameters respectively.

**Code:** The most important part of the component is the code that it executes. This code must be written in Jython but need not necessarily be completely in Python. It can make calls to Java classes and methods and thus execute Java code. The code for the new component is entered into the provided text area (Figure 11).

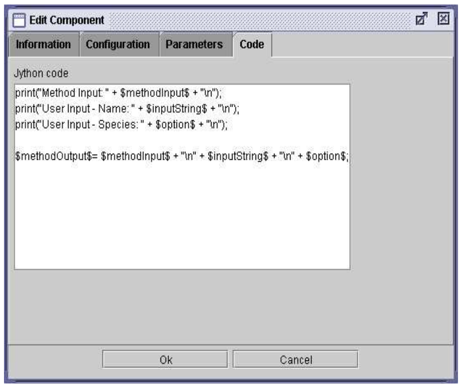
### *Placing the Component Within an Appropriate Group*

Once the user has created a new visual component, (s)he must add it within an appropriate group in the Component Palette. Note that all the components in the `HyperThesis` Component Palette are arranged into groups. Each group contains a group of components with functionality related to the subject of that group. The Component Palette in Figure 2 depicts various groups. Apart from using the existing groups within the repository, new groups can also be created.

## 3.4 Summary

In summary, some of the key advantages of using `HyperThesis` are as follows:

- The graphical interface makes constructing bioinformatic workflows fast and intuitive for biologists and bio-programmers alike. If one can visualize it then one can implement it using `HyperThesis`.
- One can expand the range of functionalities available in `HyperThesis` simply by creating user-defined workflow components and adding them to workflows. Keeping one's `HyperThesis` up-to-date is just a matter of adding new components as new algorithms develop in ones area of interest. The `HyperThesis` framework guarantees that new and old components can communicate with each other.
- Finally, the ever increasing amount of bioinformatics data can be gathered, and analyzed in a standard, integrated manner in `HyperThesis`, allowing bioinformatics researchers to concentrate on gathering and analysis of data and relieving them of the burden of learning and utilizing individual stand alone tools.



Jython code

```
print("Method Input: " + $methodInput$ + "\n");
print("User Input - Name: " + $inputString$ + "\n");
print("User Input - Species: " + $option$ + "\n");

$methodOutput$= $methodInput$ + "\n" + $inputString$ + "\n" + $option$;
```

**Figure 11:** GUI for *code* specification.

## 4. A CASE STUDY

In this section, we illustrate with an example how `HyperThesis` can be used to integrate bioinformatics applications. We first briefly introduce the *XomatiQ* component [11] of the gRNA which we are going to use in our example later.

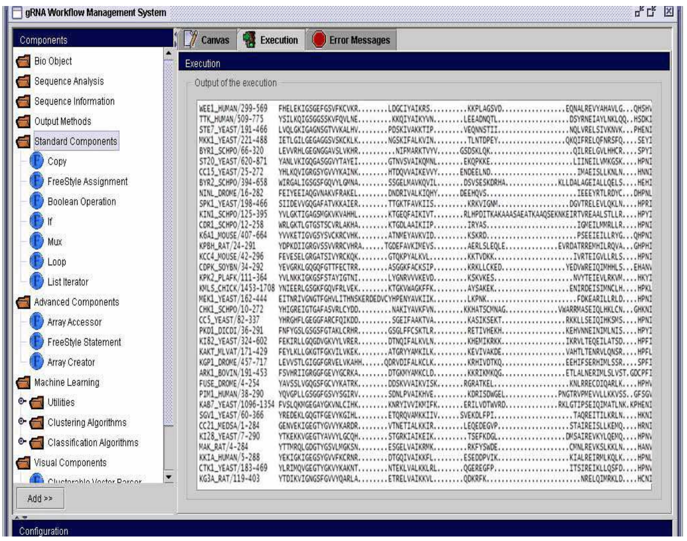
## 4.1 XomatiQ

In the gRNA, geographically distributed biological sources are converted to XML format and then stored locally in a commercial RDBMS using the *Data Hounds* component [11]. In the Data Hounds component, we first generate a relational schema. Second, we transform data from various sources to XML format by creating valid XML documents of the corresponding data. Third, we parse XML documents created from the previous step and load them into tuples of relational tables in a standard commercial DBMS.

*XomatiQ* [11] is build on top of the Data Hounds component and provides an XML query language to facilitate the querying of one or more such distributed or local warehouses managed within the gRNA. It supports a visual XML-based query interface. Through the interface, DTD structures of stored XML documents are displayed, and users can formulate queries by clicking the relevant data elements and entering conditions. Such queries are specified in a language similar to XQuery [9].

Once a user formulates a query using XomatiQ, it is transformed to corresponding one or more SQL queries over the relational generic schema. These queries are evaluated against the database where the data is stored in relational tables. Upon successful execution of the SQL queries, the results are formatted as XML documents (if necessary) and returned back to the user or passed to another application for further

```
FOR      $a IN
         document("dTrEMBL.DEFAULT")/hlx_trembl
WHERE    (($a/db_entry/keywords/keyword = "Kinase") AND
         ($a/db_entry/sequence[@length < 200]) AND
         ($a/db_entry/sequence[@length > 100]))
RETURN   {
         <MyResult>,
         $a/db_entry/sequence/text()
         </MyResult>
         }
```
**Figure 12:** Text version of the XomatiQ query.



```
WEE1_HUMAN/299-569   FHELEKIGSGEFGSVFKCVKR........LDGCIYAIKRS..........KKPLAGSVD................EQNALREVYAHAVLG...QHSHV
TTK_HUMAN/509-775    YSILKQIGSGGSSKVFQVLNE.........KKQIYAIKYVN........LEEADNQTL.................DSYRNEIAYLNKLQQ..HSDKI
STE7_YEAST/191-466   LVQLGKIGAGNSGTVVKALHV........PDSKIVAKKTIP........VEQNNSTII..................NQLVRELSIVKNVK...PHENI
MKK1_YEAST/221-488   IETLGILGEGAGGSVSKCKLK........NGSKIFALKVIN.........TLNTDPEY.................QKQIFRELQFNRSFQ....SEYI
BYR1_SCHPO/66-320    LEVVRHLGEGNGGAVSLVKHR..........NIFMARKTVYV......GSDSKLQK......................QILRELGVLHHCR....SPYI
ST20_YEAST/620-871   YANLVKIGQGASGGVYTAYEI........GTNVSVAIKQMNL.......EKQPKKE.......................LIINEILVMKGSK....HPNI
CC15_YEAST/25-272    YHLKQVIGRGSYGVVYKAINK........HTDQVVAIKEVVY......ENDEELND.......................IMAEISLLKNLN....HNNI
BYR2_SCHPO/394-658   WIRGALIGSGSFGQVYLGMNA........SSGELMAVKQVIL.......DSVSESKDRHA.............KLLDALAGEIALLQELS....HEHI
NINL_DROME/16-282    FEIYEEIAQGVNAKVFRAKEL........DNDRIVALKIQHY......DEEHQVS........................IEEEYRTLRDYC...DHPNL
SPK1_YEAST/198-466   SIIDEVVGQGAFATVKKAIER........TTGKTFAVKIIS.........KRKVIGNM..................DGVTRELEVLQKLN....HPRI
KIN1_SCHPO/125-395   YVLGKTIGAGSMGKVKVAHHL........KTGEQFAIKIVT........RLHPDITKAKAAASAEATKAAQSEKNKEIRTVREAALSTLLR....HPYI
CDR1_SCHPO/12-258    WRLGKTLGTGSTSCVRLAKHA........KTGDLAAIKIIP.........IRYAS.........................IGMEILMMRLLR....HPNI
K6A1_MOUSE/407-664   YVVKETIGVGSYSVCKRCVHK........ATNMEYAVKVID.........KSKRD.........................PSEEIEILLRYG...QHPNI
KPBH_RAT/24-291      YDPKDIIGRGVSSVVRRCVHRA......TGDEFAVKIMEVS.........AERLSLEQLE.............EVRDATRREMHILRQVA...GHPHI
KCC4_MOUSE/42-296    FEVESELGRGATSIVYRCKQK........GTQKPYALKVL..........KKTVDKK.....................IVRTEIGVLLRLS....HPNI
CDPK_SOYBN/34-292    YEVGRKLGQGQFGTTFECTRR........ASGGKFACKSIP.........KRKLLCKED................YEDVWREIQIMHHLS...EHANV
KPK2_PLAFK/111-364   YVLNKKIGKGSFSTAYIGTNI........LYGNRVVVKEVD.........KSKVKES.....................NVYTEIEVLRKVM....HKYI
KMLS_CHICK/1453-1708 YNIEERLGSGKFGQVFRLVEK........KTGKVWAGKFFK.........AYSAKEK.....................ENIRDEISIMNCLH....HPKL
MEK1_YEAST/162-444   EITNRIVGNGTFGHVLITHNSKERDEDVCYHPENYAVKIIK.........LKPNK.........................FDKEARILLRLD....HPNI
CHK1_SCHPO/10-272    YHIGREIGTGAFASVRLCYDD.........NAKIYAVKFVN........KKHATSCMNAG.............VWARRMASEIQLHKLCN...GHKNI
CC5_YEAST/82-337     YHRGHFLGEGGFARCFQIKDD.........SGEIFAAKTVA.........KASIKSEKT.................RKKLLSEIQIHKSMS....HPNI
PKD1_DICDI/36-291    FNFYGSLGSGSFGTAKLCRHR........GSGLFFCSKTLR.........RETIVHEKH.................KEHVNNEINIMLNIS....HPYI
KI82_YEAST/324-602   FEKIRLLGQGDVGKVYLVRER........DTNQIFALKVLN.........KHEMIKRKK.................IKRVLTEQEILATSD....HPFI
KAKT_MLVAT/171-429   FEYLKLLGKGTFGKVILVKEK........ATGRYYAMKILK.........KEVIVAKDE.................VAHTLTENRVLQNSR....HPFL
KGP1_DROME/457-717   LEVVSTLGIGGFGRVELVKAHH......QDRVDIFALKCLK.........KRHIVDTKQ.................EEHIFSERHIMLSSR....SPFI
ARK1_BOVIN/191-453   FSVHRIIGRGGFGEVYGCRKA........DTGKMYAMKCLD.........KKRIKMKQG.................ETLALNERIMLSLVST.GDCPFI
FUSE_DROME/4-254     YAVSSLVGQGSFGCVYKATRK........DDSKVVAIKVISK........RGRATKEL..................KNLRRECDIQARLK....HPHV
PIM1_HUMAN/38-290    YQVGPLLGSGGFGSVYSGIRV........SDNLPVAIKHVE.........KDRISDWGEL...............PNGTRVPMEVVLLKKVSS..GFSGV
KAB7_YEAST/1096-1354 FVSLQKMGEGAYGKVNLCIHK........KNRYIVVIKMIFK.......ERILVDTWVRD..............RKLGTIPSEIQIMATLNK.KPHENI
SGV1_YEAST/60-366    YREDEKLGQGTFGEVYKGIHL........ETQRQVAMKKIIV......SVEKDLFPI....................TAQREITILKRLN....HKNI
CC21_MEDSA/1-284     GENVEKIGEGTYGVVYKARDR........VTNETIALKKIR........LEQEDEGVP...................STAIREISLLKEMQ....HRNI
KI28_YEAST/7-290     YTKEKKVGEGTYAVVYLGCQH........STGRKIAIKEIK.........TSEFKDGL..................DMSAIREVKYLQEMQ....HPNV
MAK_RAT/4-284        YTTMRQLGDGTYGSVLMGKSN........ESGELVAIKRMK.........RKFYSWDE...................CMNLREVKSLKKLN....HANV
KKIA_HUMAN/5-288     YEKIGKIGEGSYGVVFKCRNR........DTGQIVAIKKFL........ESEDDPVIK...................KIALREIRMLKQLK....HPNL
CTK1_YEAST/183-469   YLRIMQVGEGTYGKVYKAKNT........NTEKLVALKKLRL.......QGEREGFP....................ITSIREIKLLQSFD....HPNV
KG3A_RAT/119-403     YTDIKVIGNGSFGVVYQARLA........ETRELVAIKKVL........QDKRFK.......................NRELQIMRKLD....HCNI
```
**Figure 13:** Results of Multiple Sequence Alignment.

processing. The reader may refer to [11] for detailed discussion on Data Hounds and XomatiQ. We now illustrate the XomatiQ component with an example.

Assume that we have warehoused TrEMBL database [7] using the Data Hounds component. The TrEMBL database contains the translations of all coding sequences (CDS) present in the EMBL Nucleotide Sequence Database, which are not yet integrated into Swiss-Prot. Suppose we wish to retrieve all the sequences in the TrEMBL database whose length is between 100 and 200 and belongs to the "kinase" family. The corresponding text format of the query is shown in Figure 12. Note that we can also evaluate queries spanning multiple databases using the XomatiQ [11].

## 4.2 Multiple Alignment of Sequences (MAS)

We now discuss a workflow created by using `HyperThesis` that demonstrates a typical procedure done by molecular biologists. Through this example we shall show how biological data sources (TrEMBL [7]) and bioinformatics tools (ClustalW [20] and HMM [13]) are integrated efficiently and intuitively using `HyperThesis`.

Multiple alignment of sequences [16], since its introduction in the early seventies, has become a cornerstone of modern molecular biology. It has traditionally been used to deduce structure/function by homology, to detect conserved motifs and in phylogenetic studies. In the most general terms, a multiple alignment is a representation of a set of related sequences, where equivalent residues are aligned either in rows, or more usually in columns.

Multiple alignments are produced by a wide variety of programs. For instance, ClustalW [20] is a very common program to find MAS. It uses a progressive alignment technique. One of the problem with this algorithm is that it is very time consuming even for a moderate number of sequences and the complexity grow's polynomially with number of sequences. Hidden Markov Modeling (HMM) is another very successful method for finding multiple sequence alignment [13]. In this method a particular HMM is trained to recognize a set of aligned sequences. Once we have this model we can align any new set of sequences against the model. The complexity of the alignment algorithm is linear with number of sequences and one can align a very large number of sequences. Hence, HMM is a better choice for aligning large number of sequences compared to ClustalW.

We now demonstrate how we can perform multiple alignment of sequences using the HMM in `HyperThesis`. Assume that we have warehoused TrEMBL database [7] using the Data Hounds component [11]. Suppose we wish to align all the sequences in TrEMBL that belongs to the "kinase" family and whose length is between 100 and 200. Figure 2 depicts the formulation of the problem using the `HyperThesis`. Let us elaborate on the semantics of this workflow. This problem can be broken down into the following two steps: (1) training the HMM model and (2) use the trained model to align new sequences. In the first step, we get a small set of sequences from a related protein family ("kinase" in our case) and save them in FASTA format ($FastaMultiSequences$ component in Figure 2). Then, the multiple alignment of these sequences is generated using ClustalW (*multiAlign* component). This set of alignments are then given to an HMM to build the model ($HMMBuild$ component) and then saved for future use ($HMMSave$ component). Note that $HMMSave$ component may not be available in the component repository. Hence, the user has to create this component (discussed in Section 3.3) to save the Hidden Markov Model into a file.

In the second step, we retrieve all the sequences in the TrEMBL database [7] whose length is between 100 and 200 and belongs to the "kinase" family using the XomatiQ (Figure 12) in the gRNA ($XomatiQ$ $Sequences$ component). This component is discussed in Section 4.1. Note that `HyperThesis` enables us to connect to the XomatiQ directly through the visual components. Then, we use the trained model to align the sequences ($HMMAlign$ component in Figure 2). Finally, we print out the results of the multiple alignment as shown in Figure 13 ($ListIterator$, $getSequence$ and *print* components).

Observe that without `HyperThesis` we need to use three different tools (ClustalW, HMM, and a data integration and querying system) with three different interfaces to perform the above task. Also, we would require to convert the output from one tool to a format acceptable as an input to the next. Indeed, `HyperThesis` allows bioinformatics researchers to concentrate on gathering and analysis of data and relieves them of the burden of learning and utilizing individual stand alone tools.

## 5. RELATED WORK

Our proposed bioinformatics applications integration system is largely influence by several recent technologies by two research communities. In one hand, the bioinformatics community has been involved in the research and development of generic biological data and applications integration system that would facilitate integration of geographically dispersed, heterogeneous, biological data sources and programs. On the other hand, the workflow community has focused on application development using processes, programs, databases, and so on. We review some of these technologies here.

Conventional workflow management systems [10] do not offer support for ad hoc integration of arbitrary data repositories, programs, and the likes. Workflow technology has been exploited in the past to develop several commercial and research prototypes with limited success. Substantial amount coding is still needed in such system to cope with changes, specially when new resources are needed to be added into the system.

Recently, several commercial biological workflow systems such as Functional Bioinformatics System [3], TurboBench Workflow [8], and *überTool* [6] has been proposed. These tools provide a graphical interface to construct bioinformatic workflows intuitively. However, it is not possible to do an in-depth comparison with `HyperThesis` as to the best of our knowledge the technical reports on these tools are not publicly available. There are also several research prototypes such as IntelliGEN [14], LABBASE [19], and BioFlow [12] that address the issue of biological applications integration.

Let us next look at these above systems from the perspective of an average biologist. The richness and complexity of the language constructs in the workflow systems such as Functional Bioinformatics System [3], TurboBench Workflow [8], IntelliGEN [14], and LABBASE [19], prevent average biologists from exploiting the strengths provided by these system, and often make it difficult to do anything without mastering the system. In other words, these systems can directly increase the productivity of a bioinformatics programmer, but they probably cannot directly increase the productivity of an average biologist. However, `HyperThesis` has been designed carefully to address this issue. It provides a user-friendly graphical interface where sophisticated programs are represented as intuitive visual components which enable biologists and bio-programmers to integrate bioinformatics tools and programs in an intuitive manner.

Let us now look at these systems from the point of efficiency and availability. Most of the above systems need to connect to the online databases and applications to perform certain tasks. These has several disadvantages such as unavailability of sources, "denial of service" attack, slow responses of sources, and errors in source data [21]. In `HyperThesis`, we warehouse the geographically dispersed remote databases locally using Data Hounds and XomatiQ [11]. Furthermore, the bioinformatics tools are provided as a large local repository of interconnectable, parameterized workflow components. Hence, such problems do not arise in `HyperThesis`.

## 6. CONCLUSIONS

We have demonstrated how the gRNA provides an efficient and systematic mechanism for integrating heterogeneous bioinformatics applications. Specifically, the `HyperThesis` component enables us to integrate diverse data sources and bioinformatics tools. It is a graphical workflow management system for genomic applications which represents sophisticated functionality as intuitive visual components. It provides a large repository of interconnectable, parameterized visual components that makes constructing bioinformatic workflows fast and intuitive for biologists and bio-programmers alike. Our system has been fully implemented using Java.

**Acknowledgment.** The authors wish to thank Dr D. T. Singh, Pedro Cruz, Reina Angelica, Yusdi Santoso and Wang Yi for their contributions in making HyperThesis a reality.

## 7. REFERENCES

[1] DALI. http://www.ebi.ac.uk/dali/.
[2] Helixense Pte Ltd. http://www.helixense.com.
[3] Functional Bioinformatics System. `http://www.ppdiscovery.com/PPD_DIS_5_1_4.htm`.
[4] Interoperable Informatics Infrastructure Consortium (I3C). http://www.i3c.org.
[5] InterProScan. http://www.ebi.ac.uk/interpro/scan.html.
[6] Science Factory. http://www.science-factory.com/index.html.
[7] TrEMBL Database. http://www.ebi.ac.uk/trembl/.
[8] TurboBench Workflow System. `http://www.turbogenomics.com/products/turbobench_example.html`.
[9] XQuery 1.0: An XML Query Language. http://www.w3.org/TR/2001/WD-xquery-20011220.
[10] Wil van der Aalst, Kees van Hee. Workflow Management: Models, Methods, and Systems. *MIT Press*, January 2002.
[11] S. S. Bhowmick, P. Cruz, A. Laud. XomatiQ: Living with Genomes, Proteomes, Relations and a Little Bit of XML. *In the Proceedings of ICDE*, Bangalore, India, 2003.
[12] Z. Guan, H. M. Jamil. Streamlining Biological Data Analysis using BioFlow. *In the Proceedings of the 3rd IEEE International Symposium on Bioinformatics and Bioengineering (BIBE 2003)*, Washington DC, March 10-12, 2003.
[13] K. Karplus, C. Barett, R. Hughey. Hidden Markov Models for Detecting Remote Protein Homologies. *Bioinformatics*, 14, 846-856, 1998.
[14] K. J. Kochut, J. Arnold, A. Sheth et al. IntelliGEN: A Distributed Workflow System for Discovering Protein-Protein Interactions. *International Journal on Distributed and Parallel Database*, 2002.
[15] A. Laud, S. S. Bhowmick, P. Cruz, D. T. Singh, G. Ramesh. The gRNA: A Highly Programmable Infrastructure for Prototyping, Developing and Deploying Genomics-Centric Applications. *VLDB*, Hong Kong, China, 2002.
[16] O. Lecompte, J. D. Thompson, F. Plewniak, J. Thierry, O. Poch. Multiple Alignment of Complete Sequences in the Post-Genomic Era. *Gene: An International Journal on Genes and Genomes*, 270 (2001) 17-30.
[17] C. A. Ouzounis, P. D. Karp. The Past, Present and Future of Genome-wide Re-annotation. *Genome Biology*, 3(2), 2002.
[18] A. C. Siepel, A. N. Tolopko, A. D. Farmer et al. An Integration Platform for Heterogeneous Bioinformatics Software Components. *IBM Systems Journal*, 40(2), pp. 570–591, 2001.
[19] L. Stein, S. Rozen, N. Goodman. Managing Laboratory Workflow with LabBase. *In Proceedings of the 1994 Conference on Computers in Medicine*, (CompMed94).
[20] J. D. Thompson, D. G. Higgins, T. J. Gibsons. CLUSTAL W: Improving the Sensitivityof Progressive Multiple Sequence Alignment Through Sequence Weighting, Position-specific Gap Penalties and Weight Matrix Choice. *Nucleic Acids Research*, 22, 4673-4680, 1994.
[21] Limsoon Wong. Technologies for Integrating Biological Data. Briefings in Bioinformatics, 3(4):389–404, 2002.