PROTEUS RTI: A FRAMEWORK FOR ON-THE-FLY INTEGRATION OF BIOMEDICAL WEB SERVICES

Shahram Ghandeharizadeh, Esam Alwagait and Sriranjan Manjunath
Computer Science Department
University of Southern California
Los Angeles, California 90089-0781

ABSTRACT: *On-the-fly integration refers to scenarios where a user, say a neuroscientist, wants to integrate a Web Service immediately after discovering it. The challenge is to significantly reduce the required information technology skills to empower the user to focus on the domain-specific problem. Proteus RTI is a first step towards addressing this challenge. It includes a graphical interface to enable a user to register and integrate Web Services together. This paper emphasizes example uses of the system from a neuroscience perspective, providing a brief overview of the system.*

Keywords: *Information Integration, Web Service, Run-time optimization, Relational algebra.*

1. INTRODUCTION

Scientists in diverse disciplines are overloaded with data due to availability of better instruments and data gathering tools [2]. A laboratory may require years to process all the hypotheses and idiosyncrasies in data collected from a few weeks of experimentation. At the same time, researchers desire to share their incremental discoveries without publishing the entire data set. A solution is to publish the functionalities pertaining to the published discoveries [4, 5].

Other scientists may discover a published functionality, examine its output and determine its relevance to their activities. A scientist may want to integrate the discovered functionality with other tools on his desktop. A challenge is how to facilitate on-the-fly integration of the published functionalities. The primary objective is to significantly reduce the required information technology skill to allow the scientists to focus on the domain-specific problems.

The ideal framework must be "what"-oriented, empowering a scientist to specify "what" functionalities should be integrated and the system resolves the details of "how". This frees the scientist to focus on a domain challenge while navigating the functionality of published data sources. Based on their domain knowledge, the scientist should be able to incorporate and discard functionalities at will. The framework should execute the compositions of a scientist quickly, empowering them to change and refine their integrations. A scientist should be able to publish an integration and share it with other colleagues using electronic means.

Web Services (WSs) hold the potential to realize the envisioned ideal framework. A Web Service (WS) is a network enabled application component with service-oriented architecture using standard interface description language and communication protocols that facilitate easy development and deployment of data intensive applications. It is an emerging technology with the following key advantages. First, a WS and its operations represent the key functionalities supported by a data source and its computations. Second, software development environments such as Microsoft's .NET and Sun's Java simplify the task of publishing a WS by automating many implementation details. Third a WS is autonomous and under complete control of the organization that implements its operations.

This raises the key question: While scientific communities such as astronomers have adopted the concept of WSs [6], why is not this concept adopted widely by all scientists? The answer is partially due to novelty of WSs. There is also a lack of a "what"-oriented framework to minimize the required programming skills to integrate WSs together, placing the concept of WSs beyond the reach of many scientists. At the time of this writing, WSs remain relatively unexplored and untested in the context of bioinformatics.

Proteus RTI is a first step towards the ideal framework. A key design decision is our use of relational algebra operators to integrate autonomous WSs. RTI consists of three components: Plan Generation Graphical User Interface (PG-GUI), a COoRDinator (CORD), and a Broker. The scientist employs PG-GUI to register WSs, drag-n-drop WSs onto a canvas and invoke their operations to see their output. A scientist may define flows between multiple WSs in order to glue them together. In Section 2, we describe several example uses to demonstrate how a user integrates WSs using Proteus. Visit http://proteusrti.usc.edu for animations demonstrating these and other examples. Section 3 presents an overview of Proteus RTI and its key components. Section 4 discusses the restrictions imposed by service providers and built-in optimization techniques to enhance response time. To elaborate, popular sites such as NCBI and Google limit the number of accesses by a client to their WS. We speculate inexpensive computers in combination with use of WS replication and partitioning will eliminate these restrictions. Brief conclusions and future research directions are presented in Section 5.

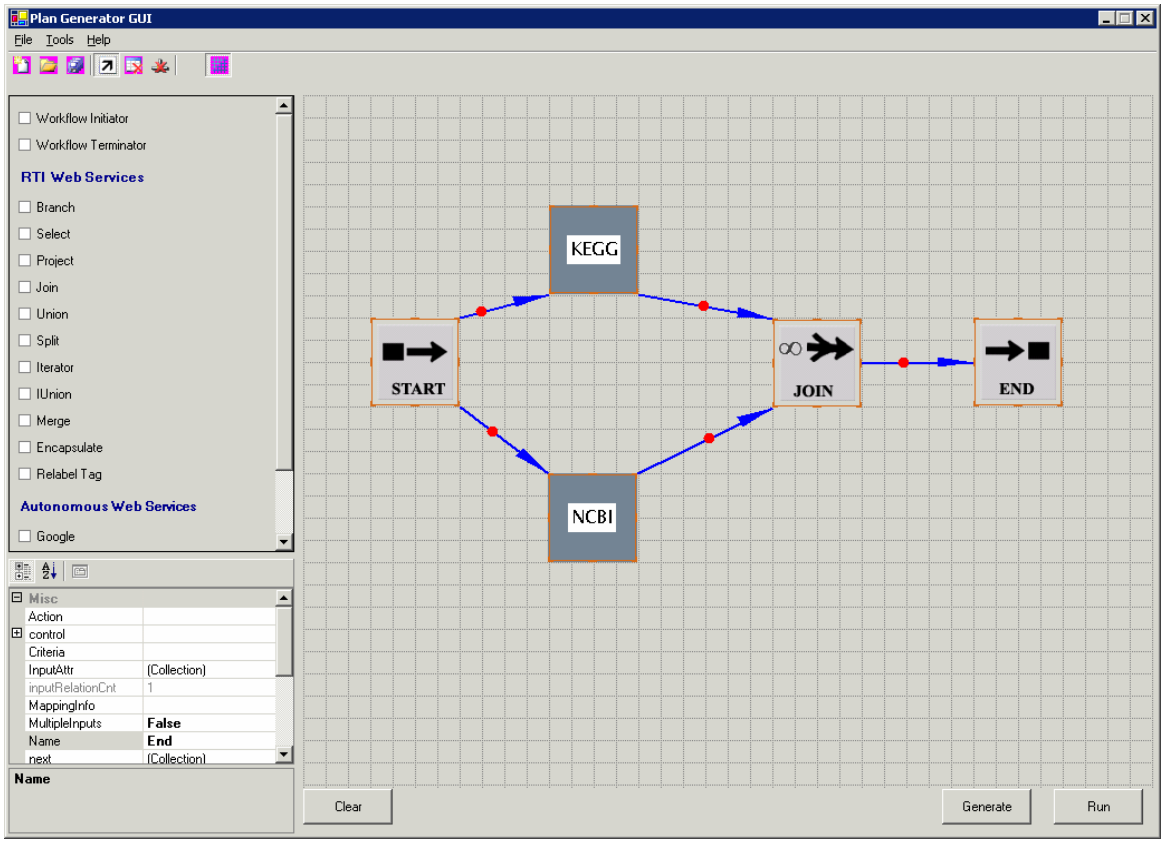

Figure 1: A query plan composing KEGG & NCBI

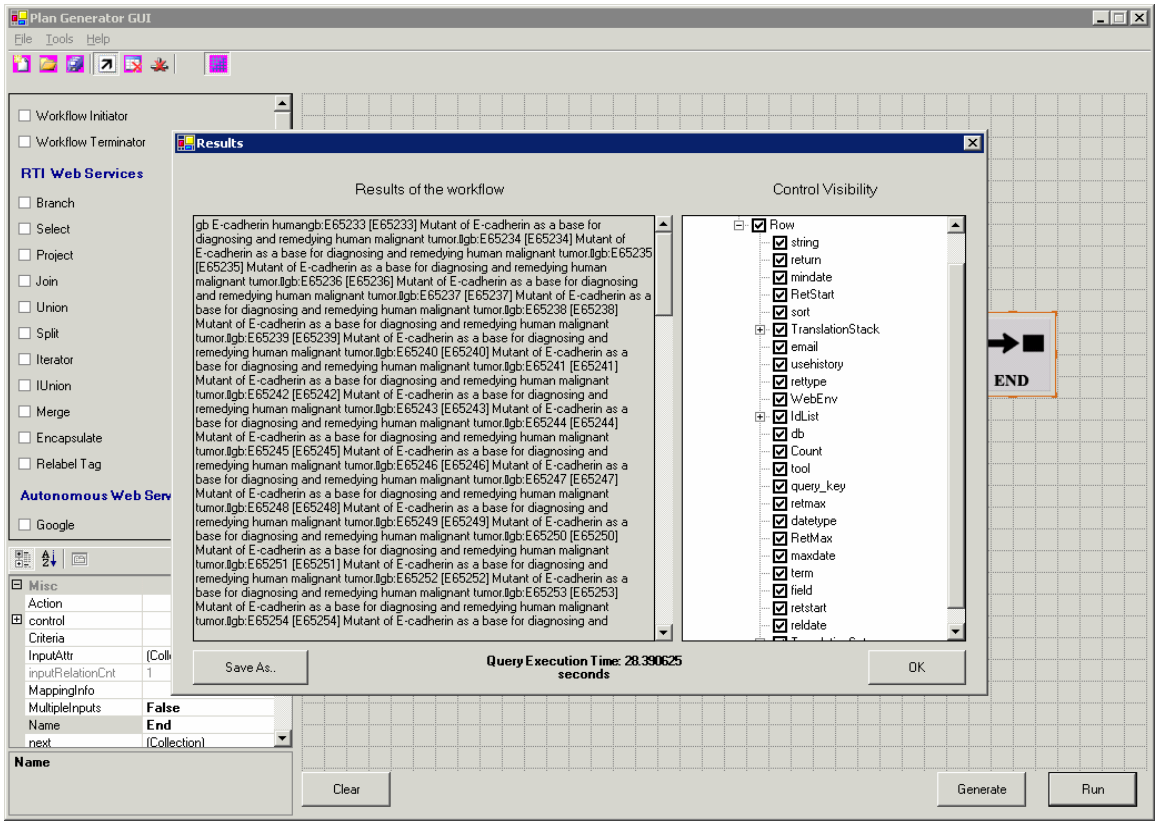

Figure 2: Result from the plan shown in Figure 1

2. A MOTIVATING EXAMPLE SCENARIO

Consider a neuroscientist who wants to retrieve information pertaining to "E-cadherin" in humans. Two relevant Web Services are KEGG and NCBI. These can be registered with PG-GUI by specifying the URL for their Web Service Description Language (WSDL). Subsequently, the user may draw a query plan such as the one shown in Figure 1. In this Figure, KEGG is encapsulated to invoke its "bfind" operation while NCBI is encapsulated to invoke its "run_eSearch_MS" operation. By encapsulating these WSs, their output is tagged[1] with the input term used to invoke each operation, namely "E-cadherin". This term is used as the join clause to bring together the outputs of NCBI and KEGG. The resulting XML output is shown in Figure 2. The right hand side shows tag names while the left hand side show tag values. One may click on the checkbox to toggle the visibility of a value for a tag name. The scientist may examine the resulting XML, save it into a file and import it into other applications that support XML. In addition, the scientist may feed the results into other WSs for further refinement and processing.

Other capabilities of the system are as follows. Using PG-GUI, a scientist may save the plan shown in Figure 1 into a file and e-mail it to a colleague who can execute the plan to retrieve its outputs. Moreover, the resulting plan can be saved into a file and published as a new WS. Using this, a scientist may construct complex plans using functionalities published by others. Of course, this does require managing the life cycle of the complex plan. This is because the simple WSs are autonomous and their publisher may change them, causing the complex plan to break. One may use PG-GUI to trouble-shoot the plan, detect those WSs that have changed, and modify the plan in response to these changes to make the complex query plan functional.

Note that by publishing a complex query plan as a WS, the user may import the new WS into PG-GUI and re-use it to author other plans.

3. OVERVIEW OF PROTEUS RTI

The example of Section 2 highlights two key design decisions. First, Proteus conceptualizes a WS as a component that consumes an XML document as input and produces an XML document as output. In its simplest form, an XML document may represent either a record or a set of records. Second, we use relational algebra operators such as the Join operator in Figure 1 to glue autonomous WSs together. Other algebraic operators used by Proteus RTI include: Select, Project, and Union. One may desire implementation of other algebraic operators such as set-intersect or set-subtract. PG-GUI is extendible and enables the user to implement the desired operators as autonomous WSs, register them with the PG-GUI, and incorporate them into a plan.

[1] See http://proteusrti.usc.edu/downloads/ncbikegg.htm for how this is accomplished.

Iterator, Branch and Split

In addition to algebraic operators, the other internal operators include Iterator, Branch, and Split. They are designed to control flow of WS invocations in a query plan and support the specification of WSs. While the Iterator operator is designed for replicated WSs, the Split operator is designed for partitioned WSs. We describe these in turn.

Iterator consumes either an XML document of elements or an array of XML elements. It produces each element and invokes the next WS with this element. It is essential because a consuming WS might accept only one XML element as its input while the producing WS generates a set of XML elements. Thus, if a WS is to be invoked with a set of XML objects, the Iterator operator is used to produce each element of the set one at a time. This operator can be implemented using either the push or the pull paradigm [3]. It is always followed by a Union operator to bring the results of an autonomous WS for a collection of input records together.

When there are multiple replicas of an autonomous WS, this operator may direct the different objects to different replicas for simultaneous processing. This implements intra-query parallelism in Proteus [1].

The Branch and Split operators denote sub-query trees that might be processed simultaneously. Each branch may execute independent of another. However, at some point, these branches must reference a common operator (either a Join or a Union) that synchronizes and merges them back into one branch. The key difference between Branch and Split is that Split is accompanied with selection clauses for each WS. To illustrate, assume a molecule document containing objects with elements: MolName and GeneSeq. With Branch, this document is duplicated and routed along each subsequent branch. With Split, each branch is provided with a selection clause, e.g., one branch is labeled with "MolName = CRH" and the other is labeled "MolName ≠ CRH". In this case, Split constructs two documents: One with "MolName = CRH" and a second with other objects. It routes each document along the specified branch. This enables Proteus to use Split to support partitioned WSs based on a range of key-values or a hash index structure. An example might be NCBI's decision to support a partitioned version of eSearch consisting of three functional WS fragments: eSearchAtoE, eSearchFtoL, eSearchLtoZ. NCBI might make this decision to improve system performance and eliminate its usage policy. In this case, Proteus uses the Split operator to direct entries to the appropriate fragment without involving the neuroscientist in this implementation detail.

Example: We now demonstrate a subset of the operators with an example query trees used in SANGAM, see http://dblab.usc.edu/Sangam. This query performs the following: For a given term find all the matching molecules with their definition and corresponding source-specific ids, showing only those with available Stressor and Brain Region data. Figure 3 shows the query tree. Its input is an XML document containing two elements: term and criterion. Examples include <cfos, protein>, <cfos, nucleotide>, and others. The Branch operator consumes this as input and

initiates two independent branches for each WS that contains relevant information: One using the KEGG FIND, and the other referencing NCBI eSearch. For each, Branch forwards its <term, criterion> input, say <cfos, nucleotide>.

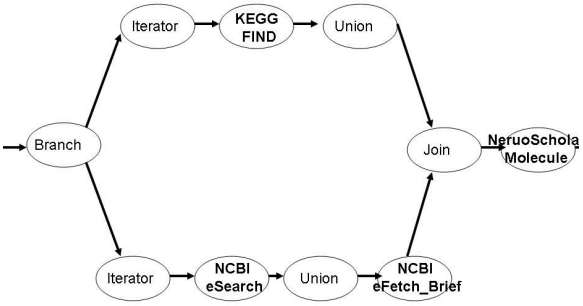

Figure 3: A Query Plan

Each branch starts with the Iterator operator. The presence of this operator depends on the assumed execution paradigm. Specifically, it is required when the query plan is executed in a distributed manner using standards such as Web Service Enhancement, WSE. To be complete, its purpose is to inform the NCBI eSearch WS to forward its results to the Union operator.

With the first branch, the Iterator invokes the Find operation of KEGG WS to retrieve matching <id, definition>. With the second branch, the Iterator invokes NCBI eSearch to obtain a list of <id> for the input <term, criterion>. The Union operator eliminates duplicate <id> values. Next, it invokes NCBI eFetch_Brief WS with one <id> at a time to retrieve its <id, definition>. Finally, the Join operator merges the output of these two branches using an equi-join on the "definition" element of each object. The resulting records are sent to NeuroScholar Molecule WS to identify those molecules with a corresponding Protein and Stressor information. The resulting information is forwarded to the GUI for display.

PG-GUI, CORD, and Broker:

Proteus RTI consists of three key components: PG-GUI, CORD, and Broker. PG-GUI is the primary interface for a user to draw query plans. CORD and Broker are in charge of efficient execution of the query plans. When the user clicks the RUN button of PG-GUI, PG-GUI requests the user to provide input to the query tree, generates the XML plan representation of the drawn query-tree with the specified input, and submits it for processing to CORD. CORD might be implemented in either a centralized or a decentralized manner. Here, we focus on its centralized implementation which walks the XML query-tree representation and its individual operators. An operator might be either an autonomous WS or an internal operator. Consider each case in turn.

With an autonomous WS, CORD invokes the Broker with the number of entries to be processed. The Broker replies with a list of replicas for the WS and the number of entries to be directed to that replica. CORD spawns a thread for each replica and provides it with the appropriate number of entries. Subsequently, CORD brings the results of these invocations together.

If an internal operator is a relational algebra operator then CORD invokes the equivalent routine that corresponds to this operator. If the internal operator is either a Branch or Split then CORD spawns a thread for each sub-query tree. With Iterator, it spawns a thread for each WS replica and passes the number of entries that should be processed by that replica. The Iterator precedes an autonomous WS, implementing the execution paradigm of the aforementioned paragraph.

4. KEY ADVANTAGES OF PROTEUS RTI

In addition to on-the-fly integration of Web Services, Proteus RTI provides the following two key advantages. First, it provides for one-time implementation of key parallelism techniques based on replication and partitioning of Web Services. Both facilitate intra and inter query parallelism. With inter-query parallelism, different WS replicas are used to process different queries. With intra-query parallelism, a single query may utilize different replicas of a WS to process multiple invocations of a query plan at the same time. One may combine fragmentation and replication by replicating popular WS fragments selectively. CORD and Broker are designed to intelligently utilize these replicated WS fragments. Using these techniques, we envision WS providers such as NCBI to lift their restrictions on the number of invocations allowed against their services. For example, NCBI may replicate those WSs that are updated infrequently using inexpensive PCs. With update intensive WSs, NCBI may partition the data set logically, e.g., WSs pertaining to data from the rat species might be assigned to a different subset of nodes than those pertaining to the human species.

Second, Proteus can be used to effectively manage the life cycle of composite WSs. When a scientist publishes a new WS referencing multiple autonomous WSs, the resulting query plan is saved. This plan may break when a service provider changes the interface to their autonomous WS. This might be due to signature mismatch or discrepancies in expected data input. One must first detect the cause of the failure in the query plan and then resolve it. PG-GUI can be used in for both steps. It can be used to recall the query plan to enable the scientist to examine its components and their output. Subsequently, the scientist may fix the query plan by either introducing new operators or changing the mapping between the input and output of the utilized WSs.

5. CONCLUDING REMARKS

Proteus RTI is a framework for a "what"-oriented interface to query Web Services and integrate them into a plan. It has many similarities to a software engineering environment for specifying flows charts. For example, similar to these tools, it can be used effectively to manage the life cycle of a project. There are also differences. Two key differences are as follows. First, participating components, i.e., WSs, are autonomous and authored by others. One may not change the operations supported by these WSs. However, one may examine their output, and extend them using both relational algebra operators and other WSs. Second, software packages

do not implement the parallelism concepts deployed by Proteus to enhance response time. Proteus can do so because it assumes the participating WSs are stateless.

We intend to extend Proteus in several directions. First, we are extending PG-GUI with a planner that composes WSs together in response to a specified desired output. The idea is as follows. A scientist would identify the WSs with relevant data and the desired output. The planner would compose a plan that integrates the identified WSs to produce the desired output. When it is impossible for the planner to compose a plan, it would present the scientist with a partial plan along with the missing data sources. Such a planner would require domain specific ontologies to realize their objective. If successful, this planner would make the system more "what"-oriented, reducing the amount of technical know-how from the user, e.g., relational algebra.

6. ACKNOWLEDGEMENTS

This research was made possible by an unrestricted cash gift from Microsoft Research and NSF research grant IIS-0307908.

References

[1] E. Alwagait, S. Ghandeharizadeh: A Comparison of Alternative Web Service Allocation and Scheduling Policies. *IEEE SCC* 2004.

[2] J. Gray, D. T. Liu, M. A. Nieto-Santisteban, A. Szalay, G. Heber, and D. DeWitt: Scientific Data Management in the Coming Decade. ACM SIGMOD Record, V 34.4, pp 35-41.

[3] G. Graefe: Query Evaluation Techniques for Large Databases. *ACM Comput. Surv*. 25(2): 73-170 (1993).

[4] S. Koslow: Discovery and Integrative Neuroscience. *Clin EEG Neurosicence*, 36(2): April 2005.

[5] R. Kötter: Neuroscience Databases: Tools for Exploring Brain Structure-Function Relationships. In *Philos Trans R Soc Lond B Biol Sci*, 356(1412), August 2001.

[6] A.S. Szalay, J. Gray, A.R. Thakar, P.Z. Kunszt, T. Malik, J. Raddick, C. Stoughton, J. vanderBerg: The SDSS skyserver: public access to the sloan digital sky server data. *ACM SIGMOD international conference on Management of data*, 570-581 (2002).