PathExplorer: Service Mining for Biological Pathways on the Web

George Zheng[1] and Athman Bouguettaya[2]

[1] Virginia Tech, Blacksburg, Virginia, USA
gzheng@vt.edu
[2] CSIRO ICT Centre, Canberra, ACT, Australia
athman.bouguettaya@csiro.au

Abstract. We propose to model biological processes using Web services to address limitations of existing biological representation methodologies. We apply our Web service mining tool, named PathExplorer, to discover potentially interesting biological pathways linking service models of biological processes. The tool uses an innovative approach to identify useful pathways based on graph-based hints and service-based simulation verifying user's hypotheses.

1 Introduction

Biological pathways are represented as networks of interactions among biological entities such as cell, DNA, RNA and enzyme. The exposure of biological pathways are expected to deepen our understanding of how diseases come about and help expedite drug discovery for treating them. Computer-based pathway study currently relies on two main approaches of entity/process representation: free-text descriptions and computer models. Free-text based approaches used in GenBank [25], DIP [28], KEGG [21, 23], Swiss-Prot [7], and COPE [20] rely on free text annotations and narratives [15, 16] to target towards human comprehension. One major disadvantage with these approaches is their inherent lack of support for computer-based simulation of these processes. Computer models (e.g., [3, 4, 17, 22, 27]) of biological processes, on the other hand, are often constructed to simulate biological processes in an isolated environment, limited to the study of known pathways, and lack the ability to facilitate the discovery of new pathways. We propose to use Web service modeling strategy [29] to bridge the gaps between the two representation approaches. Using this strategy, biological processes are modeled as Web service operations, which can be first described and published by one organization, and later *discovered* and *invoked* by other *independently* developed applications. A service operation may consume some input substance meeting a set of preconditions and then produce some output substance as a result of its invocation. Some of these input and output substances may themselves carry processes that are known to us and thus can be also modeled and deployed as Web services. Domain ontologies containing definition of various entity types would be used by these Web services when referring to their operation inputs and outputs. This service oriented process modeling and deployment strategy not only allows for the identification of pathways linking processes of biological entities, as do existing natural language processing approaches (e.g., [18, 26]), but would more importantly bring about unprecedented opportunity for validating such pathways right on the Web through direct invocation of involved services. When enough details are captured in these process models, this in-place invocation capability presents an inexpensive and accessible alternative to existing *in vitro* and/or *in vivo* exploratory mechanisms. In [31], we applied our Web service mining framework [30] to WSML/WSDL service models of biological processes that are deployed using WSMX [10] for the discovery of biological pathways. We then explored the unprecedented opportunity of evaluating such pathways right on the Web through direct invocation of involved services. In this paper, we extend our approach to also provide graph-based hints on discovered pathways to help user formulate hypotheses, which can then be either confirmed or rejected based on simulation results, leading to the identification of useful pathways.

In the remainder of this paper, we first describe in Section 2 the architecture of PathExplorer. In Section 3, we introduce how we model biological processes as WSML services. In Section 4, we describe the mechanisms PathExplorer uses to discover pathways linking WSML service models of biological processes. In particular, we focus on how PathExplorer identifies potentially interesting subgraphs within such a pathway network. In Section 5, we describe our simulation algorithm, first introduced in [31] and since extended for this paper. The algorithm is used to invoke Web services involved in discovered pathways for validation and predictive analysis purposes. We then present and discuss our simulation results from applying this algorithm. We conclude the paper in Section 6.

2 Architecture

Fig. 1 shows the architecture of PathExplorer, which starts with *scope specification*, a manual phase involving a domain expert defining mining context including functional areas (e.g., cell enzyme, drug functions) and/or locales (e.g., heart, brain) where these functions reside. Based on such mining context, PathExplorer establishes a hierarchy of domain ontology indices to speed up later phases in the mining process. Scope specification is followed by several automatic phases. The first of these is *search space determination*, where the mining context is used to define a focused library of existing Web services as the initial pool for further mining. The next is the *screening* phase, where Web services in the focused library would go through filtering algorithms for the purpose of identifying potentially interesting leads of service compositions or pathway segments. The filtering algorithms are based on the following three service/operation recognition mechanisms:

Promotion. When operation op_1 of service s_a produces an entity (i.e., output parameter) that in turn provides service s_b, we say that $s_a : op_1$ promotes s_b, as shown in Fig. 1 (a).

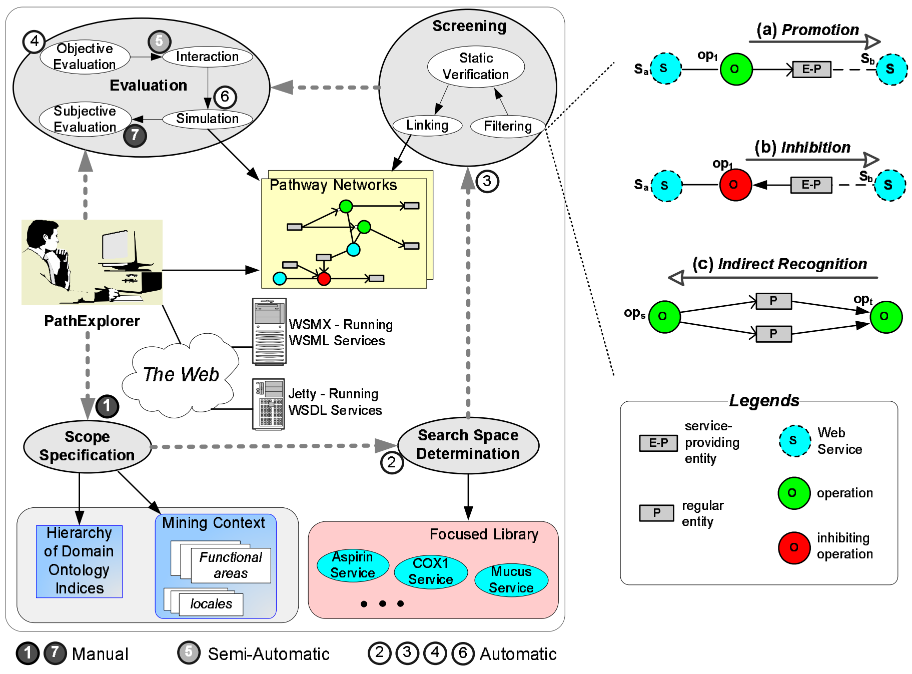


Fig. 1. PathExplorer Architecture

Inhibition. When operation op_1 of service s_a consumes an entity (i.e., input parameter) that in turn provides service s_b, we say that $s_a : op_1$ inhibits s_b as shown in Fig. 1 (b).

Indirect Recognition[3]. A *target* operation op_t indirectly recognizes a *source* operation op_s, if op_s generates some or all input parameters of op_t, as shown in Fig. 1 (c).

Based on these recognition mechanisms coupled with a publication/subscription-based algorithm [30], linkages between Web services and their operations in the focused library can be quickly established from bottom-up. These pathway segment leads are then semantically verified based on a subset of operation pre- and post-conditions involving binary variables (e.g., whether the input to an operation is activated) and enumerated properties (e.g., the locale of an operation input). Finally, verified pathway segment leads are linked together using our link algorithms [29] for establishing more comprehensive pathway network. Discovered pathways from the screening phase are input to the *evaluation* phase, which consists of four sub-phases. *Objective evaluation* identifies and highlights interesting segments of a pathway by checking whether such linkages are novel (i.e.,

[3] Indirect recognition is in contrast to direct recognition [30], where an operation can be directly invoked by another. Direct recognition is applicable to fields such as e-commerce but not pathway discovery and is thus not included here.

previously unknown). An *interactive* session follows next with the user taking hints from these highlighted interesting segments within the pathway network and picking a handful of nodes representing services, operations and parameters to pursue further. PathExplorer then attempts to link these nodes into a fully connected graph using a subset of nodes and edges in the original graph. This subgraph provides the user the basis to formulate hypotheses. As an example, such a hypothesis may state that an increase in the dosage amount of Aspirin will lead to the relief of pain, but may inadvertently increase the risk of ulcer in the stomach. These hypotheses can be tested out via *simulation*, which involves PathExplorer invoking the relevant service operations, changing the quantity/attribute value of various entities involved. Simulation results showing the dynamic relationships between these biological entities are then presented to the user, whose *subjective evaluation* finally determines whether the pathway in pursuit is actually useful.

3 Service-based Modeling of Biological Processes

To model biological processes as Web services, we first compiled a list of conceptual process models shown in Fig. 2 that are based on [2], [6], [14], [19] and [24]. In addition to describing process models, these sources also reveal some simple relevant pathways that can be manually put together. Multiple examples of *promotion*, *inhibition* and *indirect recognition* can be found in these pathways. For example, Fig. 2 (a) shows that 15 LO provides an operation called *produce LXA4*, which promotes the service of LXA4. Fig. 2 (c) shows that upon injury, LTB4 recruits Neutrophil, promoting its service of producing COX2. Fig. 2 (i) shows that Gastric Juice's service can inhibit the services of both Stomach Cell and Mucus. Examples of *indirect recognition* can be found in Fig. 2 (h), where PLA2's service can liberate Arachidonic Acid, which can in turn be used as input to either the *produce PGG2* operation of COX1's service or the *produce PGE2* operation of the COX2 service. Examples of pre- and post-conditions can be found in Fig. 2 (g), where NF-κB/Rel when not phosphorylated can translocate from cytoplasm to cell nucleus, where it can stimulate proinflammatory gene transcription. NF-κB/Rel's service, however, may be inhibited by the IκB service if NF-κB/Rel is bound by its corresponding operation when IκB is not phosphorylated. We use process models such as these as references when we develop real Web services. We also use simple pathways manually constructed here as references when we check the correctness of pathways automatically discovered in Section 4 using our mining algorithms.

Each of the conceptual process models is next captured in a Java class and exposed through Axis2 [1] running inside a Jetty Web server [5] as a WSDL service. Although the internal details of biological processes can be modeled as WSDL Web services, WSDL itself does not provide elaborate mechanism for expressing the pre- and post-conditions of service operations. WSDL also lacks the semantics needed to unambiguously describe data types used by operation input and output messages. We choose WSML [8] among others (e.g., OWL-S [13], WSDL-S [11]) to fill this gap due to the availability of WSMX, which supports

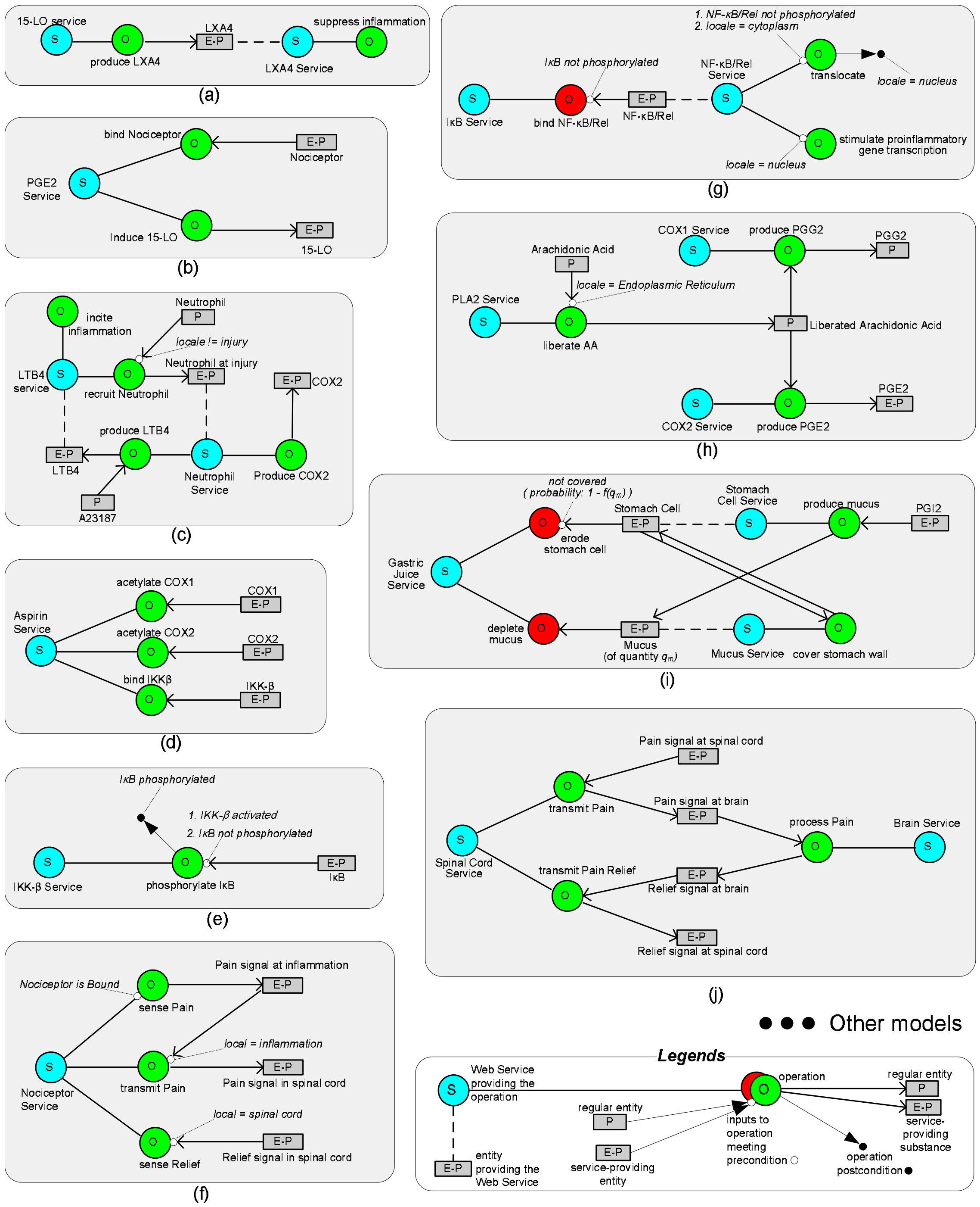


Fig. 2. Examples of Conceptual Process Model and Simple Pathway

the deployment of ontologies and Web services described in WSML. We categorize biological entities within our mining context into several ontologies. These include *Fatty Acid*, *Protein*, *Cell*, and *Drug*. They would all refer to a *Common* ontology containing generic entity types such as *Substance*, the root concept of all entity types. We use *UnknownSubstance* as a placeholder for process inputs that are not fully described in the literature. We also create a *Miscellaneous* ontology capturing definitions of entity types found in the literature that don't seem to belong to any domain. Fig. 3 shows several ontologies including those for cells, proteins, fatty acids and the miscellaneous entities rendered in WSMT [9].

Using these ontologies, we then wrap the semantic interfaces of existing WSDL services as WSML services. WSML supports the descriptions of pre- and post-conditions in the capability section and the ontological type description in the interface section. Fig. 4 gives an example of each for the *NF_kappaB_Rel* service. The capability section states for the precondition that the input entity instance named *nfkbr* should be of type *NF_kappaB_Rel* (defined in the protein ontology). In addition, *nfkbr*'s locale should be cytoplasm and it should not be phosphorylated. The interface section states that input entity *NF_kappaB_Rel* has grounding with the *translocate* operation of the corresponding WSDL service. The output from this service operation should be mapped to *NF_kappaB_Rel* as defined in the protein ontology.

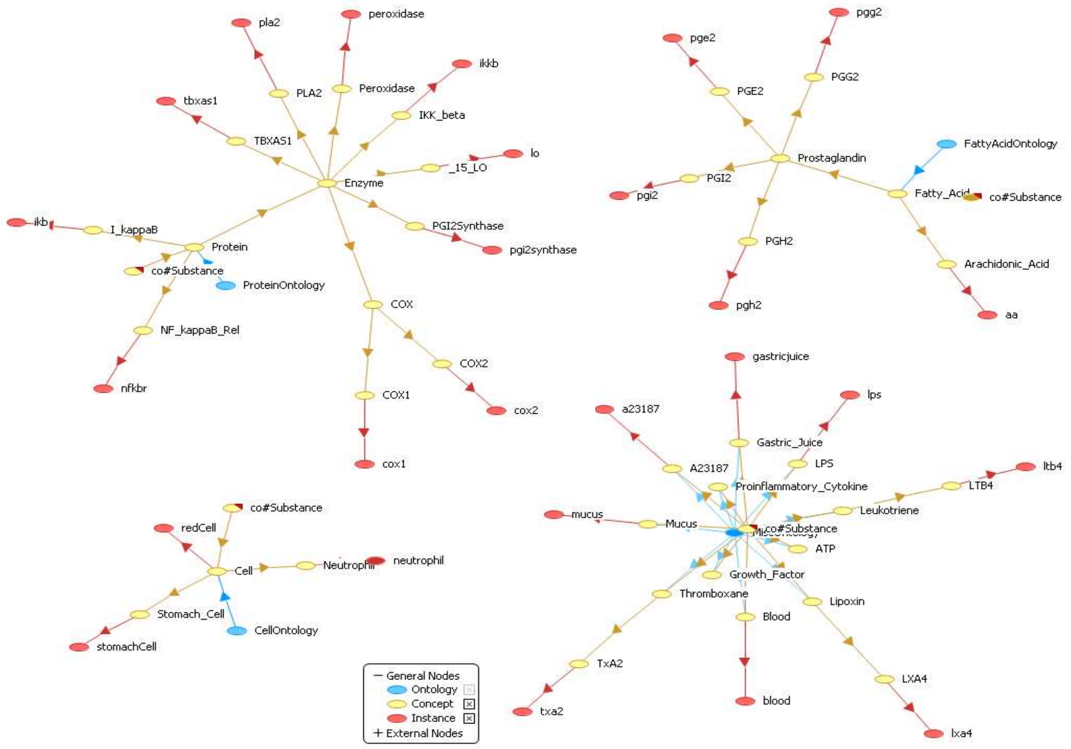
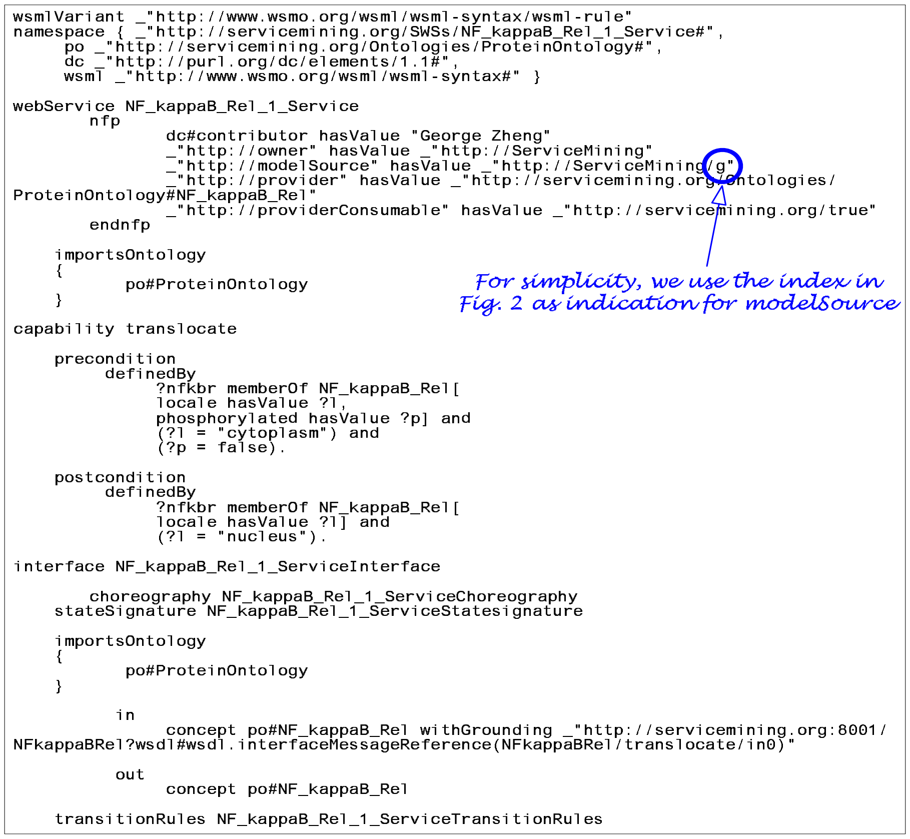


Fig. 3. Example Ontologies Rendered in WSMT



```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://servicemining.org/SWSs/NF_kappaB_Rel_1_Service#",
     po _"http://servicemining.org/Ontologies/ProteinOntology#",
     dc _"http://purl.org/dc/elements/1.1#",
     wsml _"http://www.wsmo.org/wsml/wsml-syntax#" }

webService NF_kappaB_Rel_1_Service
       nfp
              dc#contributor hasValue "George Zheng"
              _"http://owner" hasValue _"http://ServiceMining"
              _"http://modelSource" hasValue _"http://ServiceMining/g"
              _"http://provider" hasValue _"http://servicemining.org/Ontologies/
ProteinOntology#NF_kappaB_Rel"
              _"http://providerConsumable" hasValue _"http://servicemining.org/true"
       endnfp

    importsOntology
    {
           po#ProteinOntology
    }

capability translocate

    precondition
         definedBy
              ?nfkbr memberOf NF_kappaB_Rel[
              locale hasValue ?l,
              phosphorylated hasValue ?p] and
              (?l = "cytoplasm") and
              (?p = false).

    postcondition
         definedBy
              ?nfkbr memberOf NF_kappaB_Rel[
              locale hasValue ?l] and
              (?l = "nucleus").

interface NF_kappaB_Rel_1_ServiceInterface

       choreography NF_kappaB_Rel_1_ServiceChoreography
    stateSignature NF_kappaB_Rel_1_ServiceStatesignature

    importsOntology
    {
           po#ProteinOntology
    }

          in
              concept po#NF_kappaB_Rel withGrounding _"http://servicemining.org:8001/
NFkappaBRel?wsdl#wsdl.interfaceMessageReference(NFkappaBRel/translocate/in0)"

          out
              concept po#NF_kappaB_Rel

    transitionRules NF_kappaB_Rel_1_ServiceTransitionRules
```

For simplicity, we use the index in Fig. 2 as indication for modelSource

Fig. 4. Semantic Interface Description in WSML

To work with WSML, we have made slight adaptations to our screening algorithms so they can be applied directly to WSML services. First, we add a *provider* property in the non functional properties (nfp) section of each WSML service to indicate the corresponding ontological type of an entity that can provide the service. PathExplorer uses this information to establish the relationship between a service providing entity and the service it provides. Second, we add a *modelSource* property in the nfp section to indicate the source information that the model is based on. Third, we add a *providerConsumable* property in the nfp section to indicate to PathExplorer whether the service providing entity should be consumed along the invocation of its operation. For example, in order for mucus (Fig. 2 (i)) to cover the wall of stomach, the mucus itself will have to be consumed. Finally, our validation algorithm has been customized to work with the service interrogation APIs of the WSMX runtime library for determining the overlap between the postcondition of a source operation and the precondition of a target operation. Unfortunately, WSML allows for the specification of pre- and post-conditions for only an entire service, but not its individual operations. Thus we have to split services that each originally has multiple operations into several services (e.g., *NF_kappaB_Rel_1_Service* and *NF_kappaB_Rel_2_Service*) so that different conditions can be individually specified for these operations.

PathExplorer uses the name of these services to keep track of their relationship and uses that information to merge these services towards the end of the screening phase. During simulation, PathExplorer uses lowering/lifting adapters [31] to convert ontological entity instances used by WSML services to/from SOAP messages used by WSDL services.

4 Discovery of Interesting Pathways

Algorithms from [29] and [31] enabled PathExplorer to discover potentially interesting pathways linking WSML service models of biological processes as represented in Fig. 2. To support pathway visualization, PathExplorer generates a *GraphML* file for each discovered pathway network and uses yEd [12] to render the corresponding graph. We have since also developed new algorithms in PathExplorer to help user, during the evaluation phase, formulate hypotheses that would lead to the identification of useful pathways extended from interesting segments. The addition of the *modelSource* property (Fig. 4) in the nfp section allows PathExplorer to identify novel (i.e., interesting) linkages between service models in a discovered pathway network by comparing the source indicator of linkages in the pathway graph representing the three types of service/operation recognitions as shown in Fig. 1. PathExplorer then automatically highlights these edges in the graph, presenting them as visual aid to the user for focusing more on nodes that may lead to the identification of useful pathways. After the user selects nodes of interest, PathExplorer attempts to link them into a fully connected graph to the extent possible using the following steps:

- Coalescing nodes (e.g., a, b, c in Fig. 5) linked by interesting edges into a group.
- Converting interesting nodes (e.g., t picked by user) and groups encompassing interesting nodes (e.g., c, f) into nuclei, i.e., graph expansion focus nodes.

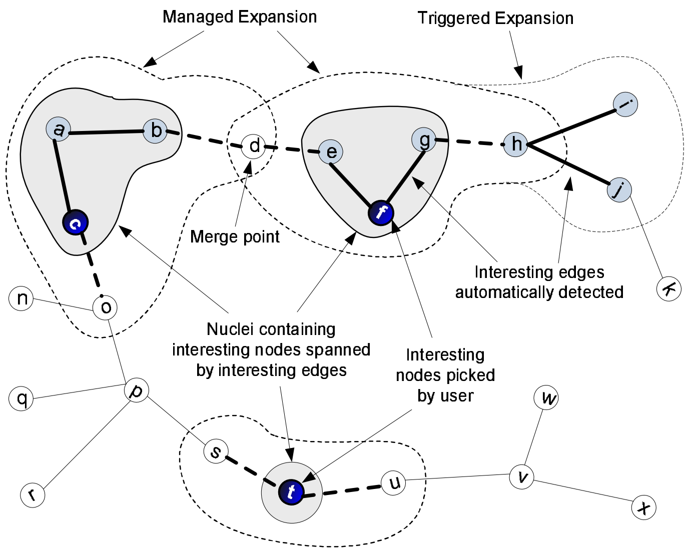

Managed Expansion
Triggered Expansion
a
b
c
d
e
f
g
h
i
j
k
n
o
p
q
r
s
t
u
v
w
x
Merge point
Interesting edges automatically detected
Nuclei containing interesting nodes spanned by interesting edges
Interesting nodes picked by user

Fig. 5. Expansion of Interesting Segments

– Incrementally expanding all the nuclei. We use the heuristics of connecting all the interesting nodes using as many interesting edges as possible. To achieve this, whenever a newly encountered node is part of a non-nucleus group (e.g., one that contains h, i and j), an additional expansion is also triggered and the whole group are engulfed. The expansion stops when all nuclei are connected or when all nodes in the graph are visited.

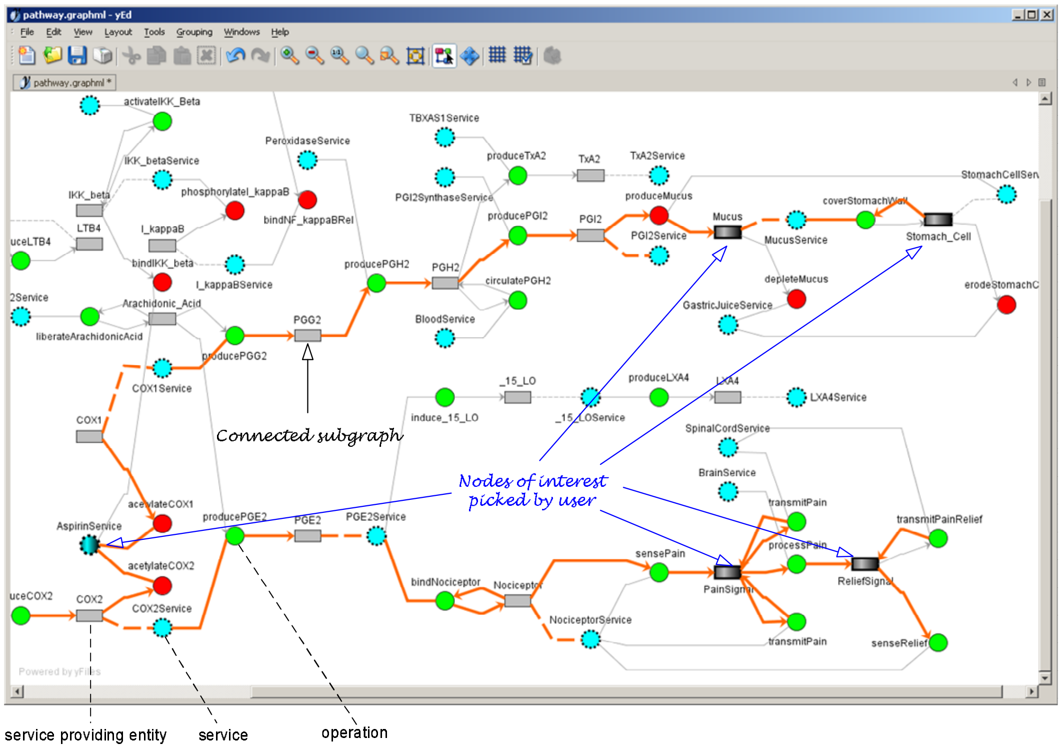


Fig. 6. Discovered Pathway Rendered in yEd

Discovered pathway networks are first presented to the user in yEd graphs with interesting linkages highlighted. Once interesting nodes are picked by the user, PathExplorer attempts to use the above process to link them into a fully connected subgraph, an example of which is shown in Fig. 6 and highlighted with thick edges. Such graphs are then presented to the user as basis for hypothesis formulation. For brevity, we display only shortened names for nodes in the graph. We keep the full name containing either the ontological path for entity nodes or the WSML service path for both service and operation nodes in a separate description field (not shown in Fig. 6). In addition, we omit pre- and post-condition details of operation linking edges such as the two forming a loop between operation *coverStomachWall* and entity *Stomach_Cell* [4]. However,

[4] The precondition along the upper edge states that Stomach_Cell is not covered by Mucus and the postcondition along the lower edge states that Stomach_Cell is covered by Mucus.

we keep track of the pre- and post-conditions in our algorithm as such information along with the ontological entity paths and WSML service paths are needed when we try to invoke these services during simulation. To ensure the correctness of our algorithms, we compared segments within the automatically discovered pathway network with those constructed manually in Fig. 2 and found them to be consistent in all cases.

5 Pathway Simulation

The verification of pathway validity can be carried out using a simulation environment, where functions of biological Web services can be invoked in the order as identified in pathway leads. A pathway lead identified in the screening phase indicates the potential possibility of a pathway based on service and operation recognition. Verification aims at determining if segments of an identified pathway lead can indeed be enabled with a chain of relevant conditions. The second important aspect of runtime simulation is its ability to support predictive analysis. Based on pathway leads established from the screening phase and later highlighted in the interactive hypothesis formulation sub-phase, the user may attempt to predict certain outcome from indirect relationships derived from the way the pathway network is laid out. For example, the user may predict based on Fig. 6 that an increase in the dosage amount of Aspirin will lead to the relief of pain, but may also increase the risk of ulcer in the stomach. Such prediction can be tested out using a simulation strategy outlined in Algorithm 1.

When an operation is to be invoked, the algorithm checks two factors. First, it examines whether all the pre-conditions of the operation are met. An operation that does not have available input entities meeting its preconditions should simply not be invoked. Second, it determines how many instances are available for providing the corresponding service. This factor is needed due to the fact that biological entities of the same type *each* has a discrete service process that deals with input and output of a finite proportion. The available instances of a particular service providing entity will drive the amount of various other entities they may consume and/or produce. For this reason, the algorithm treats each entity node in a pathway network such as one shown in Fig. 6 as a container of entity instances of the noted ontology type. In some cases, the service provider is also used as an input parameter. For example, the *sensePain* operation from the *NociceptorService* in Fig. 2 (f) has a precondition stating that the *Nociceptor* itself should be bound in order to provide this service. In order to express this precondition, we decided to include the service providing entity also as an input parameter. In cases such as this, the number of service providing instances will be determined by checking further whether each of the service providing entity instances also meets the precondition of the corresponding operation.

In Algorithm 1, an initial number of instances for each entity type et are first generated based on function $f(et)$ (lines 01-03). It is conceivable that an expert may want to create different number of instances at the beginning for different entity types. Next, we conduct I iterations of operation invocations (lines 05-31). We take a snapshot of the quantities at the end of each iteration

Algorithm 1 Simulation Algorithm

Input: Pathway Network PN, function $f()$ determining initial number of instances for an entity type, total number of iterations I, upper bound S for random number generator *random* with uniform distribution
Output: Statistics *Stats*
Variables: entity type *et*, entity instance container *Container(et)* of type *et*, operation *op*, input entity op_{in}, output entity op_{out} and precondition op_{pre}

```
1:  for all $et \in PN$ do
2:      $Container(et) \leftarrow$ create $f(et)$ instances;
3:  end for
4:  $Stats \leftarrow$ Tally entity quantities in each container;
5:  for $i = 0$ to $I$ do
6:      for all $op \in PN$ do
7:          $s \leftarrow op.getProviderServce()$;
8:          $et_{parameter} \leftarrow op.getInputParameter().getEntityType()$;
9:          $et_{provider} \leftarrow s.getProviderEntityType()$;
10:         if $et_{parameter} = et_{provider}$ then
11:             $n \leftarrow$ number of entities of type $et_{provider}$ that match $op_{pre}$
12:         else
13:             $n \leftarrow$ number of entities of type $et_{parameter}$
14:         end if
15:         $n \leftarrow n/S + ((random.nextInt(S) < (n\ modulo\ S))?1 : 0)$;
16:         for $j = 0$ to $n$ do
17:             if $\exists op_{in} \in Container(et_{parameter}) : op_{in}\ matches\ op_{pre}$ then
18:                 $op_{out} \leftarrow invoke(op)\ with\ op_{in}$;
19:                 if $et_{parameter} \neq et_{provider} \wedge$ provider is consumable then
20:                     $Container(et_{provider}).remove(0)$;
21:                 end if
22:                 if $et_{parameter} \neq et_{provider} \vee$ provider is consumable then
23:                     $Container(et_{parameter}).remove(op_{in})$;
24:                 end if
25:                 $et_{parameter} \leftarrow op_{out}.getEntityType()$;
26:                 $Container(et_{parameter}).add(op_{out})$;
27:             end if
28:         end for
29:     end for
30:     $Stats \leftarrow$ Tally entity quantities in each container;
31: end for
```

and before the very first iteration (lines 30 and 04). We determine the number of times the corresponding operation should be invoked based on the quantity of the corresponding service providing entity (lines 7 to 15). To make sure that an operation from a service providing entity of a small quantity also gets the chance to be invoked, a random number generator is used (line 15). Upon invocation of the operation, we remove corresponding entity instance based on the truth table depicted in Fig. 7. When we determine the provider should be removed (lines 19 to 21), we remove the first instance found in the corresponding container. Since the provider is not the input parameter, it is consequently not involved in the evaluation of the operation precondition. Thus we can remove any one instance found in the container. Lines 22 to 24 are for removing the input parameter instance when the corresponding condition is met. Finally, we add the output parameter instance to the corresponding entity container (lines 25 and 26).

Simulation results obtained from each run by the PathExplorer are compiled into an Excel spreadsheet, which is then used to generate a plot such as those in Fig. 8 (for the graph in Fig. 6), where the horizontal axis is for the number of iterations and vertical axis is for quantity. Fig. 8 (a) shows that when the quantity

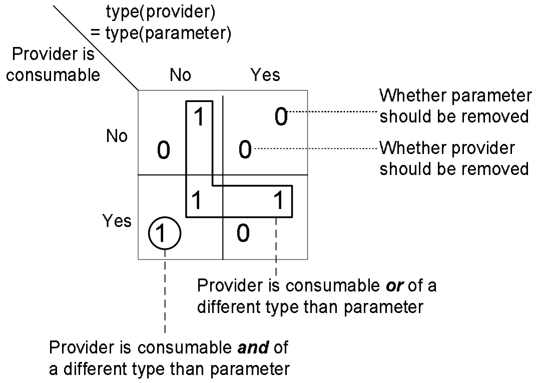

type(provider)
= type(parameter)
Provider is consumable
No
Yes
No
Yes
1
0
0
0
1
1
1
0
Whether parameter should be removed
Whether provider should be removed
Provider is consumable ***or*** of a different type than parameter
Provider is consumable ***and*** of a different type than parameter

Fig. 7. Logic for Removing Entity Instance After Operation Invocation

of Aspirin is 10, there is no sign of stomach erosion. When the quantity of Aspirin increases to 40 in Fig. 8 (b), the quantity of stomach cell drops to around 30 after 150 iterations of operation invocation. This confirms the user hypothesis that Aspirin has a side effect on the stomach. In addition, we also noticed that given a fixed quantity of Aspirin, the reduction of the initial quantity of *COX1* also has a negative effect on the stomach (Fig. 8 (c) and (d)). When the initial quantity of *COX1* is high, it takes longer for all the *COX1* to get acetylated by Aspirin. As a result, enough *PGG2* and consequently *PGH2* and *PGI2* will be built up to feed into the *produceMucus* operation of the *StomachCellService*. As the initial quantity of *COX1* becomes smaller and while the depletion rate of *Mucus* by *GastricJuiceService* remains the same, less *Mucus* is being produced by the *StomachCellService* as less *PGI2* becomes available.

While Figures 8 (a) to (d) clearly illustrate the relationships between Aspirin and *Stomach_Cell*, the relationship between the dosage amount of Aspirin and the sensation of pain is less obvious in these Figures. Except for Fig. 8 (a), which shows some accumulation of *PainSignal* when the quantity of Aspirin is 10, the rest of plots show no pattern of such accumulation or the variation thereof. A closer look at the highlighted pathway in Fig. 6 reveals that this is actually consistent with the way the simulation is set up. Since *PainSignal* is created and then converted by the *Brain* to *ReliefSignal*, which disappears after it is sensed by *Nociceptor*, this whole path at the bottom actually acts as a 'leaky bucket'. To examine exactly what is going on along that path, we decided to make two changes in the simulation setting. First, we reduce the maximum frequency of invoking the *Brain* service to half that of *Nociceptor*. This creates a potential imbalance between the production rate of *PainSignal* and *ReliefSignal* since the *processPain* operation from the *BrainService* will be consequently invoked less frequently than the *sensePain* operation from the *NociceptorService*. Second, we disable the *senseRelief* operation of the *NociceptorService*. This essentially stops the leaking of the *ReliefSignal* that are generated as a result of the *PainSignal*. When we apply only the first change to the simulation, the imbalance of the processing rates for *PainSignal* and *ReliefSignal* results in a net accumulation of *PainSignal* when the quantity of Aspirin is 10 (Fig. 8 (e)). When the quantity

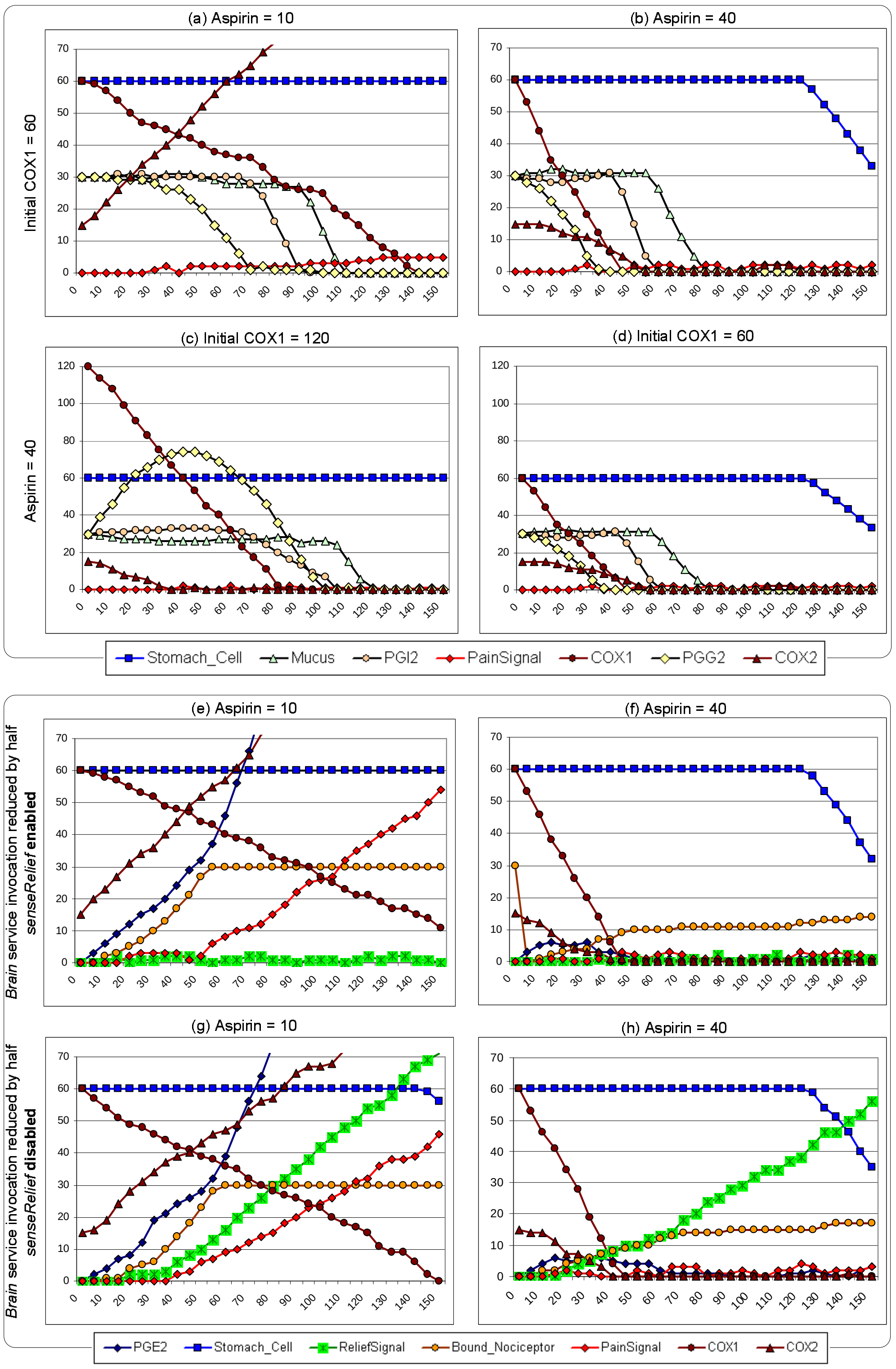


Fig. 8. Simulation Results

is increased to 40 (Fig. 8 (f)), we see there are some occasional and temporary accumulation of *PainSignal*. Finally, we apply the second change along with the first one. Consequently, we notice that while the pattern of *PainSignal*'s accumulation hasn't changed much, there is a consistent accumulation of *ReliefSignal*. Since each *PainSignal* is eventually converted to a *ReliefSignal* by the *Brain* according to the highlighted pathway in Fig. 6, the rate of *ReliefSignal*'s accumulation actually provides a much better picture on how fast *PainSignal* has been generated. We see that as the dosage amount of Aspirin increases, less *ReliefSignal* is generated, an indication that less *PainSignal* has been generated. Thus it is obvious that the increase of the dosage amount of Aspirin has a positive effect on the suppression of *PainSignal*'s generation. This confirms the other half of user's original hypothesis.

Simulation results such as these presented in Fig. 8 provide information to a pathway analyst who would otherwise get such information from *in vitro* and/or *in vivo* exploratory mechanisms. Using the service-oriented simulation environment, the interrelationships among various entities involved in the pathway network can now be exposed in a more holistic fashion than traditional text-based pathway discovery mechanisms, which inherently lack the simulation capability.

6 Conclusion

We proposed to model biological processes as Web service to bridge the gap between free-text description and traditional computer models of these processes. We presented our service mining tool named PathExplorer and demonstrated the feasibility of applying our service mining strategy to the discovery of pathways linking service models of biological processes. We described how PathExplorer identifies interesting segments in a pathway graph and automatically establishes a fully connected graph linking nodes that the user is interested in exploring. The graph, which is highlighted inside the discovered pathway network provides the user the basis for formulating hypothesis, which can then be tested out through simulation.

References

1. Apache axis2/java - next generation web services. http://ws.apache.org/axis2/.
2. Aspirin. http://www3.interscience.wiley.com:8100/legacy/college/boyer/0471661791/cutting_edge/aspirin/aspirin.htm.
3. Bps: Biochemical pathway simulator. http://www.brc.dcs.gla.ac.uk/projects/bps/.
4. Copasi. http://mendes.vbi.vt.edu/tiki-index.php?page=COPASI.
5. Jetty. http://www.mortbay.org/.
6. Nf-kappab pathway. http://www.cellsignal.com/reference/pathway/NF_kappaB.jsp.
7. Uniprotkb/swiss-prot. http://www.ebi.ac.uk/swissprot/.
8. The web service modeling language wsml. http://www.wsmo.org/wsml/wsml-syntax.
9. The web service modeling toolkit (wsmt). http://sourceforge.net/projects/wsmt.
10. Web services execution environment. http://sourceforge.net/projects/wsmx.

11. Web services semantics - wsdl-s. http://www.w3.org/Submission/WSDL-S/.
12. yed - java graph editor. http://www.yworks.com/en/products_yed_about.htm.
13. Owl-s: Semantic markup for web services. November 2004. http://www.w3.org/Submission/OWL-S/.
14. Sunny Y. Auyang. From experience to design - the science behind aspirin. http://www.creatingtechnology.org/biomed/aspirin.htm.
15. Roger Brent and Jehoshua Bruck. Can computers help to explain biology? *Nature*, 440(23):416 – 417, March 2006.
16. Jacques Cohen. Bioinformatics: An introduction for computer scientists. *ACM Computing Surveys*, 36(2):122 – 158, 2004.
17. H. de Jong and M. Page. Qualitative simulation of large and complex genetic regulatory systems. In *Proceedings of the 14th European Conference on Artificial Intelligence, ECAI*, pages 141 – 145, Amsterdam, 2000.
18. Daming Yao et. al. Pathwayfinder: Paving the way toward automatic pathway extraction. In *Proceedings of the Second Conference on Asia-Pacific Bioinformatics*, volume 29, pages 52 – 62, Dunedin, New Zealand, 2004. Australian Computer Society, Inc.
19. Craig Freudenrich. How pain works. http://health.howstuffworks.com/pain.htm.
20. Horst Ibelgaufts. Cope - cytokines online pathfinder encyclopaedia. http://www.copewithcytokines.de/.
21. M. Kanehisa, S. Goto, M. Hattori, K. F. Aoki-Kinoshita, M. Itoh, S. Kawashima, T. Katayama, M. arki, and M. Hirakawa. From genomics to chemical genomics: new developments in kegg. *Mucleic Acids Research*, 34:354 – 357, January 2006.
22. Peter D. Karp, Suzanne Paley, and Pedro Romero. The pathway tools software. *Bioinformatics*, 18:S1 – S8, 2002.
23. Kanehisa Laboratories. Kegg: Kyoto encyclopedia of genes and genomes. http://www.genome.jp/kegg/.
24. Misia Landau. Inflammatory villain turns do-gooder. http://focus.hms.harvard.edu/2001/Aug10_2001/immunology.html.
25. NCBI. Genbank. http://www.ncbi.nlm.nih.gov/Genbank/.
26. See-Kiong Ng and Marie Wong. Toward routine automatic pathway discovery from on-line scientific text abstracts. volume 10, pages 104 – 112, 1999.
27. Masaru Tomita, Kenta Hashimoto, Koichi Takahashi, Thomas Simon Shimizu, Yuri Matsuzaki, Fumihiko Miyoshi, K. Saito, S. Tanida, Katsuyuki Yugi, J. C. Venter, and C. A. Hutchison III. E-cell: software environment for whole-cell simulation. *Bioinformatics*, 15(1):72–84, 1999.
28. UCLA. Database of interacting proteins. http://dip.doe-mbi.ucla.edu/.
29. George Zheng and Athman Bouguettaya. Mining web services for pathway discovery. In *2007 VLDB Workshop on Data Mining in Bioinformatics*, Vienna, Austria, September 2007.
30. George Zheng and Athman Bouguettaya. A web service mining framework. In *2007 IEEE International Conference on Web Services (ICWS)*, Salt Lake City, Utah, July 2007.
31. George Zheng and Athman Bouguettaya. Discovering pathways of service oriented biological processes. In *The Ninth International Conference on Web information Systems Engineering (WISE 2008)*, Auckland, New Zealand, September 2008.