A Process Catalog for Workflow Generation

Michael Wolverton, David Martin, Ian Harrison and Jerome Thomere

SRI International
333 Ravenswood Ave
Menlo Park, CA 94025
<lastname>@ai.sri.com

Abstract. As AI developers increasingly look to workflow technologies to perform complex integrations of individual software components, there is a growing need for the workflow systems to have expressive descriptions of those components. They must know more than just the types of a component's inputs and outputs; instead, they need detailed characterizations that allow them to make fine-grained distinctions between candidate components and between candidate workflows. This paper describes PROCAT, an implemented ontology-based catalog for components, conceptualized as *processes*, that captures and communicates this detailed information. PROCAT is built on a layered representation that allows reasoning about processes at varying levels of abstraction, from qualitative constraints reflecting preconditions and effects, to quantitative predictions about output data and performance. PROCAT employs Semantic Web technologies RDF, OWL, and SPARQL, and builds on Semantic Web services research. We describe PROCAT'S approach to representing and answering queries about processes, discuss some early experiments evaluating the quantitative predictions, and report on our experience using PROCAT in a system producing workflows for intelligence analysis.

Recent research and development in technology for intelligence analysis has produced a large number of tools, each of which addresses some aspect of the *link analysis problem*—the challenge of finding events, entities, and connections of interest in large relational data sets. Software developed in recent projects performs many diverse functions within link analysis, including detecting predefined patterns [1–4], learning these patterns of interest [5], classifying individuals according to group membership [6] or level of threat [7], resolving aliases for individuals [8], identifying neighborhoods of interest within the data, and others.

While these tools often perform complementary functions within the overall link analysis space, there has been limited success in getting them to work together. One-time integration efforts have been time-consuming to engineer, and lack flexibility. To address this problem, a recent focus of research has been to link these tools together dynamically, through workflow software [9], a blackboard system [10], or some other intelligent system architecture.

One key challenge in building this kind of dynamic link analysis workflow environment is representing the behavior of the individual link analysis processes being composed. In this paper, we describe an implemented Process Catalog software component—PROCAT for short—that serves information about processes that allows a

workflow generation component to select, rank, and execute them within a workflow. (Our focus here is on PROCAT'S design and functionality; space constraints allow for only a few brief comments about the characteristics of the larger workflow system.) PROCAT is based on a layered approach to process representation, in which a process is described in terms of both the qualitative features that distinguish it from other processes, and quantitative models that produce predictions of the process's outputs and performance.

PROCAT is implemented and deployed within the TANGRAM workflow architecture, a complex system that generates and executes workflows for intelligence analysis. This deployment requires it to be integrated with several other workflow modules developed by other contractors. PROCAT's current knowledge base encodes a collection of real link analysis tools that perform a variety of functions. It produces quantitative predictions of those tools that early experiments suggest are accurate.

The sections that follow describe PROCAT and its use in more detail. First, we give an overview of its approach and architecture. We then describe the *Capabilities Layer* of the process description, which represents processes at a qualitative level. Next, we cover the *quantitative layers* of the process description, including some experiments evaluating the accuracy of those layers' predictions. We then discuss PROCAT's use within the TANGRAM workflow system in some detail. And finally, we compare this work to other related research, and outline future work and other research issues.

1 Overview

The problem of designing a process characterization language for link analysis presents a number of research challenges, many of which arise because of the need for flexibility in the process description. The representation must be flexible to accommodate heterogeneous processes, multiple possible workflow systems that reason about processes at differing levels of fidelity, and the evolution of the workflow systems' reasoning abilities as they are developed over time.

To meet this need of flexibility, PROCAT is built upon a layered approach to process characterization, where each process is represented at multiple levels of fidelity, and where the workflow system can retrieve and reason about processes at the representation level(s) they can handle. The layers include

- *Capabilities*, which provides a qualitative description of the process's behavior, along with the hard constraints for running it
- *Data Modification*, which describes how the statistical profile of the data (e.g., the number of nodes and links of each type) changes as the process is run over it
- *Performance*, which quantitatively describes the performance of processes (e.g., time to complete, maximum amount of memory consumed) given a data set with a particular statistical profile
- *Accuracy*, which describes how the accuracy of each node and link type in the data changes as the process is run over it

The Capabilities Layer is described in the next section, while the Data Modification and Performance Layers are described in the subsequent, Quantitative Predictions, section. While PROCAT's current process representation incorporates the Accuracy Layer,

models for producing accuracy predictions are not currently part of the system and are part of our ongoing work.

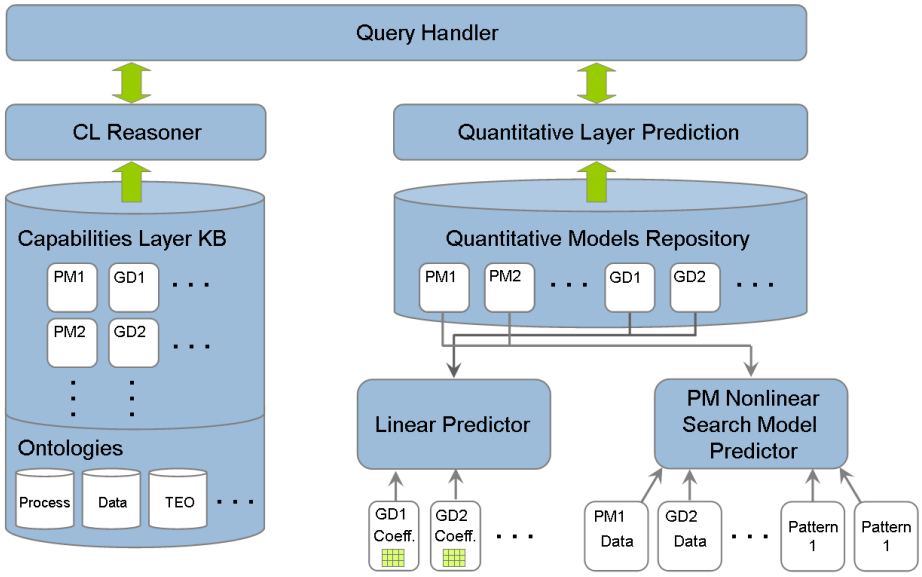


Fig. 1. PROCAT Architecture. The Query Handler accepts a variety of queries about processes from an external Workflow Component (not shown in the figure), and farms these queries out to the appropriate reasoning module.

The PROCAT architecture is shown in Figure 1. PROCAT feeds information about processes to the workflow system via a set of predefined query types. The system has two mechanisms for answering these queries. Queries that involve the Capabilities Layer are answered by reasoning over a set of ontologies that encode the processes' functionality, resource requirements, invocation details, and so on. Queries that involve the quantitative layers are answered using the processes' models from the Quantitative Models Repository. These two mechanisms are described in more detail in the next two sections.

2 Capabilities Layer

The Capabilities Layer (CL) describes in a qualitative, symbolic manner what a process does, what the requirements are for running it, and how it is invoked. This section explains the representational approach and ontology underlying PROCAT's Capabilities Layer, and the manner in which capabilities queries are handled.

2.1 Capabilities Ontology

PROCAT employs the Resource Description Framework (RDF) [11], the Web Ontology Language (OWL) [12], and the RDF query language SPARQL [13]. Each of these knowledge representation technologies has been standardized at the World Wide Web Consortium in recent years, as part of the Semantic Web initiative. A number of syntaxes have been defined for OWL, which is layered on top of RDF. PROCAT makes use of the RDF/XML syntax [14], as discussed below. The internal representation of RDF and OWL content takes the form of a set of *triples*, which are maintained in a *triple store*. OWL, a description logic language, is well suited to the Capability Layer's objectives of describing, classifying, and answering queries about categories of processes, individual processes, and their characteristics.

In PROCAT, a *process* is any well-defined, reusable procedure, and a *process installation* is an executable embodiment of a process. In the TANGRAM application, the processes described in PROCAT are data analysis programs, and each process installation is a version of a program installed on a particular machine. The capabilities ontology is organized around the PROCESS class. That class can very naturally be decomposed into a hierarchy of categories of processes for various purposes. TANGRAM, as shown in Figure 2, employs a hierarchy of data analysis processes. Some TANGRAM queries quite naturally need to ask for processes belonging to a particular subclass within this hierarchy, with additional query constraints expressed using properties. We refer to the set of terms defined in the capabilities ontology, along with certain conventions for its use, as the Process Description Language (PDL).

Fig. 2. Class Hierarchy of Processes for TANGRAM

The core of a capabilities description is a functional characterization of the process, in terms of its parameters (inputs and outputs), preconditions that must be met to run it, and postconditions that will hold after running it. The most essential element of parameter characterization is *type*. The type of each parameter is specified as an OWL

class or an XML datatype. A parameter also has a *role* (e.g., *HypothesisOutputRole* in Figure 5); roles may be shared across multiple processes. The characterization of a parameter also includes such things as default values and invocation conventions. Figure 3 shows the main classes that are found in the process ontology, and relationships between them. Properties HASINPUT and HASOUTPUT relate a process to instances of class PARAMETER. In general, those instances will also be instances of data source classes (classes that indicate the types of parameters). Thus, multiple inheritance is used to indicate parameter types, which simplifies in some ways the expression of both descriptions and queries. Some conditions may be represented using properties of these data source classes. For instance, some data analysis processes take an analysis pattern as one of their inputs.

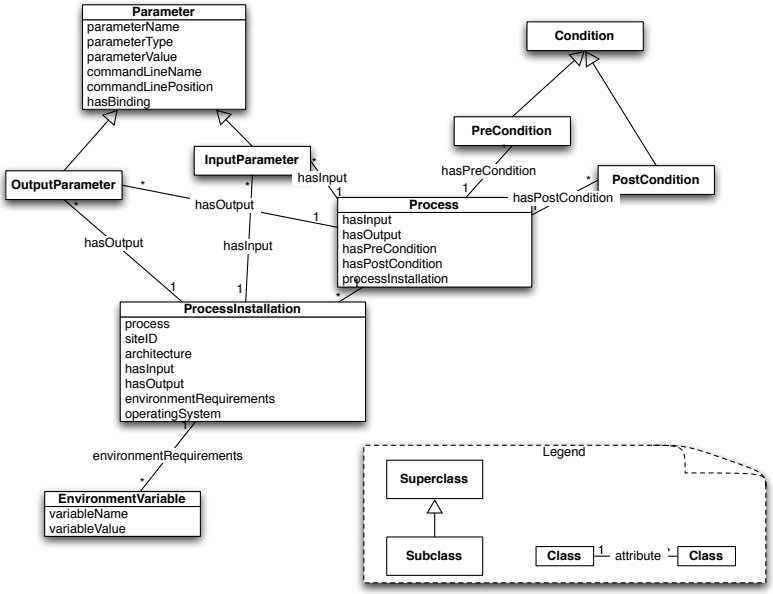


Fig. 3. Key Classes and Relations in the Process Ontology

A process may be related to zero or more instances of PROCESSINSTALLATION, each representing a specific installation of a process on a particular machine. Various characteristics are specified for process installations, including resource requirements and invocation details. HASINPUT and HASOUTPUT are also present for PROCESSINSTALLATIONS, to allow for the specification of platform-specific parameters. Information about environment variables required to run the process is also included, as well as details about the physical characteristics of the installed platform, such as machine architecture, operating system, and other details needed to reason about execution requirements and to remotely invoke the process installation.

Preconditions and postconditions. Preconditions are conditions that must be true in order for a process to successfully occur; postconditions are conditions that will hold true after an occurrence of a process. Preconditions and postconditions in general can be difficult to represent, and can require considerable expressiveness. For one thing, they are not ordinary facts about the process. They are conditions that might or might not hold true when the process is used. The sense of a precondition is that if it evaluates true prior to an *occurrence* (execution) of the process, then (assuming the process executes normally without exceptions) that occurrence will be successful. Similarly, postconditions cannot be understood as ordinary facts about a process. Thus, in a complete logical theory of processes, neither preconditions nor postconditions could be simply asserted, but would need to be reified in some way, and subject to special handling during query answering. Further, as noted just above, preconditions and postconditions apply to process occurrences rather than processes. If both processes and process occurrences are to be explicitly included in a representational framework, it becomes necessary to explicitly capture the relationship between them, as axioms. Such axioms, however, would exceed the expressiveness of RDF and OWL, and thus increase the complexity of reasoning involved in answering queries.

In PROCAT, we have mitigated these difficulties by adopting a simplifying perspective, that a process description is considered to be a *snapshot of an arbitrary successful occurrence of a process*. (Roughly speaking, then, a process description is somewhat like a skolem representative of all possible successful occurrences of the process.) Further, because PROCAT does not store information about *actual* occurrences of a process, there is no inconsistency in including in its description facts that apply to the process itself (rather than to any particular occurrence). Adopting this perspective removes the need to explicitly distinguish between processes and process occurrences. Instead, a process description in the catalog can be viewed as capturing aspects of both at once. Given this perspective, reification is no longer needed and preconditions can be stated as facts about inputs (and postconditions as facts about outputs). For example, Figure 5, in the "PROCAT Implementation and Use" section, shows a query that will match against a process having an output dataset that is *saturatedWith* instances of the class *MoneyLaunderingEvent*. (*saturatedWith* is an ontology term meaning that as many instances as possible have been inferred within a given dataset. In the example, the query uses *?dataVariable5* to stand for the output dataset parameter.) A KB statement matching the query triple (*?dataVariable5 saturatedWith MoneyLaunderingEvent*) would be a simple example of a postcondition.

2.2 Capabilities Layer Functionality

As shown in Figure 1, the Capabilities Layer makes use of the Query Handler component and the CL Reasoning component, which in turn accesses the Capabilities Layer KB. The reasoning component includes both application-specific and general-purpose query processing functionality.

Query Handler. Queries are received by means of a Web service API, as discussed in "PROCAT Implementation and Use" below. To provide catalog services for TANGRAM, PROCAT queries and responses are expressed in a slightly extended form of the RDF/XML syntax. The extension allows for the use of variables, in a manner similar

to that of SPARQL. In our syntax, variables can appear in subject, predicate, or object position of any query triple, are named by URIs (and thus belong to a namespace), and are indicated by a question mark as the first character of the name part of the URI. (See Figure 5 for a simple example query using this syntax.) Our experience to date indicates that standard RDF/XML parsers recognize these URIs without difficulty.

SPARQL is used *internally* within PROCAT for accessing the KB, as discussed below. Although we considered using SPARQL as the external query syntax for TANGRAM, we determined that SPARQL was not well suited to meet certain application-specific requirements of TANGRAM. However, we plan to support SPARQL in future versions, as a general-purpose query syntax to supplement the existing, application-specific query conventions.

The Query Handler component is responsible for parsing the RDF/XML-based query syntax and capturing it in an internal format. Because the query syntax is consistent with RDF/XML, we are able to take advantage of standard parsing functionality. Each incoming query is parsed directly into a temporary triple store. In this way, we are able to use triple store queries and triple store manipulations in analyzing and processing the query.

Capabilities Reasoner. An important requirement for PROCAT is to provide flexibility in supporting application-specific query requirements, that is, requirements that cannot be met by a standard query language such as SPARQL. For example, in TANGRAM, one type of query asks PROCAT to formulate a commandline for a particular invocation of a given process installation. Although the KB contains the essential information such as commandline keywords, default values, and proper ordering of commandline arguments, nevertheless the precise formulation of a commandline requires the coding of some procedural knowledge that cannot readily be captured in a KB. ("PROCAT Implementation and Use" below discusses further the types of queries used in TANGRAM.)

PROCAT's architecture allows for the use of arbitrary Lisp code to provide the application-specific query processing. This code, in turn, can use a variety of mechanisms (including SPARQL queries) to examine the temporary triple store containing the parsed form of the incoming query. In most cases, this examination results in the construction of one or more queries (which, again, may be SPARQL queries) to be submitted to the capabilities KB to retrieve the needed information about the process(es) in question. Once that information has been retrieved, Lisp code is called to analyze it and formulate the requested response.

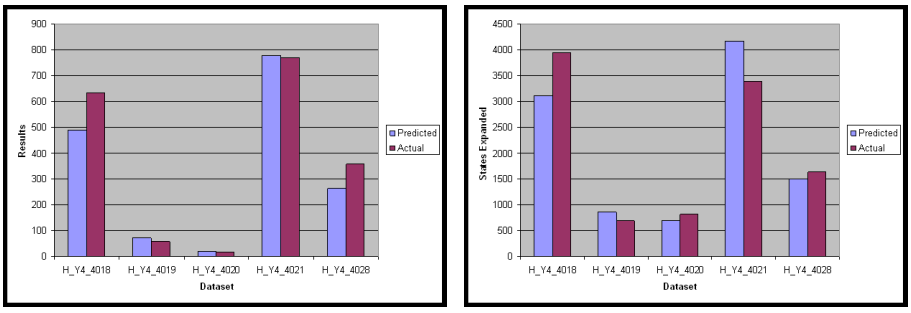
The capabilities KB, including ontologies, is stored within a single triple store. Access to the capabilities KB is provided by a layer of general-purpose (application-independent) utility procedures for triple store update, management, and querying. This includes procedures for formulating and running SPARQL queries.

3 Quantitative Predictions

We designed the quantitative prediction models of PROCAT to meet three criteria:

- The prediction models are *precise*, in that they allow fine-grained predictions of process performance.

- The prediction models are *efficient*, so that predicting a process's performance on a given data set generally takes less time than running the process.
- The individual prediction models are *composable*, so that a workflow component can accurately predict and reason about combinations of processes run in sequence or in parallel.



(a) (b)

Fig. 4. Predicted vs. actual (a) data modification and (b) performance for a pattern matching process

PROCAT currently produces two types of quantitative predictions: Data Modification and Performance.

For predictions in the Data Modification Layer, each process is described as a function that maps a problem description and a data model into a data model. The problem description will vary depending on the type of process; for example, a problem for a graph matcher could consist of a pattern graph and various parameters specifying the match criteria. A data model is a statistical description of the data set along whatever parameters are determined to be useful in selecting and composing processes. PROCAT's current data model is a statistical description of the data that consists of (1) the number of nodes of each type and (2) the branching factor per node and link type. Here, (2) is the answer to the question: For each entity type T and link type L, given a node of type T, how many expected links of type L are connected to it?[1]

Performance Layer predictions estimate the process's efficiency given the data set and problem. These predictions map a (problem description, data model, resource model) triple into a performance model. The resource model represents the performance characteristics of the hardware available to the process—processor speed, amount of phys-

[1] This level of representation implicitly assumes that link and node type distributions are independent—that is, that the existence of link L_1 attached to node T_1 tells nothing about the likelihood of link L_2 also being attached to T_1. For many processes, especially pattern matching processes, this seems to be a reasonable assumption. However, for other processes, especially relational classification processes, the independence assumption may be too strong.

ical and virtual memory, network throughput, database response, and so on. The performance model will represent the predicted time to process the data set, and possibly other measures of performance that we determine are useful for selecting processes. For example, one could imagine building a more complicated model of performance for an anytime process, which includes a tradeoff between execution time and the number of (and completeness of) the results produced.

The quantitative models are represented procedurally—as lisp functions. A model can be custom-built for a particular process by the knowledge engineer populating the catalog, or it can be created by instantiating a preexisting model class. Currently, PROCAT has two built-in model classes, shown in the lower right portion of Figure 1. The first is a linear model, where predictions of a quantity characterizing data or performance are derived via a weighted sum of features of the input data.[2] The coefficients for this model can be learned through a regression method, such as least-squares, based on runs of the process being modeled. The second is a nonlinear model that is specific to pattern matchers. This model predicts pattern matcher output and performance by using a recurrence relation to estimate the number of states expanded in the search for a match.

While a detailed description of this latter model is beyond the scope of this paper, we show in Figure 4 the results of some of its predictions for the LAW pattern matcher [4] against actual behavior of the system, to give a sense of the level of accuracy of predictions for one well-understood tool. Figure 4(a) shows the predicted and actual number of results found by LAW matching a relatively simple pattern against five different data sets varying in size, branching factor, and other characteristics. Figure 4(b) shows predicted and actual search states expanded (here we use states expanded as a proxy for runtime because of the ability to get consistent results across machines) for the same runs. The average error of the predictions was 20% for data modification, and 19% for performance. The time taken to run the prediction model was faster than the run of the pattern matcher by over two orders of magnitude.

While this model was built specifically for LAW, the data modification portion of it should be transferable to other pattern matchers. Furthermore, experiments with another pattern matcher, CADRE [3], indicate that its runtimes are roughly proportional to LAW's when matching the same pattern, despite the fact that their pattern-matching approaches are quite different. This suggests that this predictive model may be applicable (with some fitting) to other pattern matchers. Testing that hypothesis is part of our ongoing work.

4 PROCAT Implementation and Use

As discussed above, PROCAT is presently being applied as a module in the TANGRAM system for building and executing workflows for intelligence analysis. To support this application, we worked with the developers of the workflow generation/execution software, WINGS and PEGASUS [9], to design a set of queries that provide the information

[2] The linear coefficients also depend on the values of non-data parameters being passed to the process, and, for performance, the hardware on which the process is to be run.

WINGS/PEGASUS needs to instantiate and execute workflows effectively. These queries are broken into two distinct phases of workflow generation:

- In the *Backward Sweep*, WINGS produces candidate workflows starting with the desired output. Given an output data requirement (a postcondition) and class of processes, PROCAT returns all matching processes that can satisfy that postcondition, along with their input data requirements and other preconditions for running them.
- In the *Forward Sweep*, WINGS and PEGASUS prune the workflow candidates generated in the Backward Sweep, rank them, and map processes to actual grid clusters to run. The queries in this sweep require PROCAT to predict output data characteristics, predict process performance on particular Grid clusters, return actual physical location(s) of the process, return resource requirements for running the process installation, and return the relevant information for building a valid command line for the process installation.

The implementation choices for PROCAT were driven by the requirements described above, together with the fact that PDL is encoded in OWL. We decided to use ALLEGROGRAPH[3], which is a modern, high-performance, persistent, disk-based RDF graph database. ALLEGROGRAPH provides a variety of query and reasoning capabilities over the RDF database, including SPARQL, HTTP, and PROLOG query APIs and built-in RDFS++ reasoning. ALLEGROGRAPH also allows one to define a SOAP Web service API to an ALLEGROGRAPH database, including the ability to generate WSDL files from the code's SOAP server definition, to facilitate the creation of Web service clients.

PROCAT is implemented as an HTTP/SOAP-based Web service with an ALLEGROGRAPH RDF triple store. Component (process) definitions, encoded using PDL, are stored in the triple store, and SOAP services provide the specific information required by WINGS and PEGASUS. Each service is associated with a message handler and response generation code for the SOAP API. No Web service for updating KB content was developed for the initial version of PROCAT, as this capability was not essential to test its ability to provide a useful service as part of a workflow generation/execution experiment. We plan to add this service for the next release of PROCAT.

Figure 5 shows an example of a SOAP query for the Backward Sweep phase. The query specifies a general class of component (in this case, *PatternMatchingProcess*) and a requirement that the component has an output of type *Hypothesis* containing objects of type *MoneyLaunderingEvent*. PROCAT will return any actual components in its repository that match these constraints. It should be noted that all components in the repository are defined as belonging to one or more classes of component, and that these are drawn from the process ontology described above. The results are returned as RDF/XML fragments, one for each matching component instance.

5 Related Work and Discussion

The problem of process characterization and the related problem of process selection to meet a particular set of requirements have been investigated for several decades under

[3] http://agraph.franz.com/allegrograph/

```
<SOAP-ENV:Envelope>
<SOAP-ENV:Body>
<pcat:FindInputDataRequirements>
 <pcat:component xsi:type='xsd:string'>
  <rdf:RDF>
   <rdf:Description rdf:about="http://...#?component2">
    <rdf:type rdf:resource="http://.../Process.owl#PatternMatchingProcess"/>
    <pdl:hasOutput rdf:resource="http://...#?dataVariable5"/>
    <pdl:hasInput rdf:resource="http://...#?dataVariable4"/>
    <pdl:hasInput rdf:resource="http://...#?dataVariable3"/>
   </rdf:Description>
  </rdf:RDF>
 </pcat:component>
 <pcat:constraints xsi:type='xsd:string'>
  <rdf:RDF>
   <rdf:Description rdf:about="http://...#?dataVariable5">
    <pdl:hasRole
     rdf:resource="http://.../Process.owl#HypothesisOutputRole"/>
    <rdf:type rdf:resource="http://...#Hypothesis"/>
    <pdl:saturatedWith
     rdf:resource="http://...#MoneyLaunderingEvent"/>
   </rdf:Description>
  </rdf:RDF>
 </pcat:constraints>
</pcat:FindInputDataRequirements>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fig. 5. Example PROCAT Query

various research headings, including program verification, deductive program synthesis, automatic programming, AI planning, agent-based systems, Semantic Web / Grid services, and e-science. Because of space constraints, we can only mention examples from the last three of these areas. For a more extensive summary of related work, see [15].

The field of agent-based systems (ABS) includes a significant body of work on characterizing and reasoning about agent capabilities, which often are conceived as remotely invocable processes. As in earlier work on AI planning, the common denominator of many approaches is the representation of preconditions and effects, often with additional information about the inputs and outputs of the operations that an agent provides. LARKS [16], for example, employs *InConstraints* and *OutConstraints*, expressed as Horn clauses referring to inputs and outputs, respectively, for this purpose. This approach, while flexible, requires special handling for these Horn clauses outside of the description logic framework that underlies LARKS's ontology. PROCAT's approach, in contrast, remains within the representational framework defined by RDF and OWL. Agent systems have also experimented with the use of additional kinds of information, such as quality of service, response time, and other kinds of performance characterization. Generally speaking, however, these have been captured using static, one-size-fits-all characterizations, rather than computed on-the-fly based on the specifics of input datasets and resources in the execution environment, as PROCAT does.

ABS has explored a variety of styles of matchmaking. For example, in the Open Agent Architecture (OAA), [17], the basic capability description is a logic programming predicate structure (which may be partially instantiated), and matchmaking is

based on unification of goals with these predicate structures. In addition, both goals and capabilities declarations may be accompanied by a variety of parameters that modify the behavior of the matchmaking routines. Although PROCAT does not make use of unification, it achieves greater flexibility by building on SPARQL, and a more effective means of categorization of capabilities (processes) based on OWL class hierarchies.

Most recently, these same problems have been the focus of inquiry in the context of research on Semantic Web Services (SWS). This field aims to enable the automation of the development and use of Web services. The first challenge in SWS has been the enrichment of service descriptions, which essentially is the same problem as process characterization. OWL for Services (OWL-S) [18], the pioneering effort in this field, introduces the expression of preconditions and effects in a Semantic Web-compatible manner, and also relies on the ability to use OWL to construct class hierarchies of services. PROCAT's ontology is based in part on OWL-S, but goes much further in distinguishing process installations and characterizing their resource requirements and invocation methods. OWL-S also includes a composite process structure model—a set of ontology elements used to formulate a step-by-step representation of the structure of a composite process. PROCAT thus far has had no need for this kind of representation.

The Semantic Web Services Framework (SWSF) [19] builds out from OWL-S by including some additional concepts (especially in the area of messaging); employing first-order logic, which is more expressive than OWL; and drawing on the axiomatization of processes embodied in the Process Specification Language (PSL). The Web Services Modeling Ontology (WSMO) [20], is an EU-funded effort with many of the same objectives and approaches as OWL-S and SWSF. WSMO distinguishes two types of preconditions (called *assumptions* and *preconditions*), and two types of postconditions (called *postconditions* and *effects*). In addition, WSMO associates services with goals that they can satisfy, and models choreography—the pattern of messages flowing between two interacting processes. PROCAT thus far has not encountered requirements for capturing these additional aspects of processes, but could be extended in these directions if needed. On the other hand, PROCAT's expressiveness requirements have deliberately been kept smaller than those of SWSF and WSMO, allowing for relatively lightweight implementation, scalability, and quick response times.

In the field of *e-science*, scientific experiments are managed using distributed workflows. These workflows allow scientists to automate the steps to go from raw datasets to scientific results. The ultimate goal is to allow scientists to compose, execute, monitor, and rerun large-scale data-intensive and compute-intensive scientific workflows. For example, the NSF-funded KEPLER project[4] has developed an open-source scientific workflow system that allows scientists to design scientific workflows and execute them efficiently using emerging Grid-based approaches to distributed computation. Compared to PROCAT, the KEPLER Actor repository can be seen to be a more general-purpose repository for storing workflow components—both the actual software, as well as metadata descriptions of that software. However, compared to PDL, the actor definitions are impoverished, and cover simply I/O parameters.

By and large, process characterization in all these disciplines has been predominantly concerned with what we here call the Capabilities Layer of description. Where

[4] http://kepler-project.org

quantitative descriptions have been used, they have most often been used to solve very specialized problems. Further, quantitative descriptions have rarely taken advantage of probabilistic methods to characterize data modification and accuracy, or to enable predictions regarding the content and structure of generated data, as has been done in PROCAT. Another significant difference is PROCAT's combined specification of a process's behavioral characterization with the fine-grained characterization of its data products.

Looking ahead, our research directions are focused on four major areas:

- Extending the Capabilities Layer to make more sophisticated use of rules and deduction in finding matching components and inferring the requirements for running them.
- Gathering data and running more experiments to test the accuracy of PROCAT's quantitative predictions, both to evaluate the existing models and to drive the creation of new and better ones.
- Implementing the Accuracy Layer, discussed above. This layer will produce estimates (rough to begin with) of the accuracy of the output data produced by a process.
- Automating some parts of the population of the process repository. For example, we have started to automate some of the experiments needed to create the quantitative models of the Data Modification and Performance layers.

Workflow generation and execution technologies are becoming increasingly important in the building of large integrated systems. More expressive process descriptions, and new kinds of reasoning about them, will play a critical role in achieving this long-term goal. We have described the process representation and reasoning approaches embodied in PROCAT, the rationale behind its current design, its role in a particular integrated system, and research directions under investigation in connection with this work.

Acknowledgments

This research was supported under Air Force Research Laboratory (AFRL) contract number FA8750-06-C-0214. The concept of a process catalog for Tangram is due to Eric Rickard. Thanks also to the builders of the other Tangram modules discussed in this paper, including (but not limited to) Fotis Barlos, Ewa Deelman, Yolanda Gil, Dan Hunter, Jihie Kim, Jeff Kudrick, Sandeep Maripuri, Gaurang Mehta, Scott Morales, Varun Ratnakar, Manoj Srivastava, and Karan Vahi.

References

1. Boner, C.: Novel, complementary technologies for detecting threat activities within massive amounts of transactional data. In: Proceedings of the International Conference on Intelligence Analysis. (2005)
2. Coffman, T., Greenblatt, S., Marcus, S.: Graph-based technologies for intelligence analysis. Communications of the ACM **47**(3) (2004)
3. Pioch, N.J., Hunter, D., White, J.V., Kao, A., Bostwick, D., Jones, E.K.: Multi-hypothesis abductive reasoning for link discovery. In: Proceedings of KDD-2004. (2004)

4. Wolverton, M., Berry, P., Harrison, I., Lowrance, J., Morley, D., Rodriguez, A., Ruspini, E., Thomere, J.: LAW: A workbench for approximate pattern matching in relational data. In: The Fifteenth Innovative Applications of Artificial Intelligence Conference (IAAI-03). (2003)
5. Holder, L., Cook, D., Coble, J., Mukherjee, M.: Graph-based relational learning with application to security. Fundamenta Informaticae **66**(1–2) (2005)
6. Adibi, J., Chalupsky, H.: Scalable group detection via a mutual information model. In: Proceedings of the First International Conference on Intelligence Analysis (IA-2005). (2005)
7. Macskassy, S.A., Provost, F.: Suspicion scoring based on guilt-by-association, collective inference, and focused data access. In: Proceedings of the NAACSOS Conference. (2005)
8. Davis, J., Dutra, I., Page, D., Costa, V.S.: Establishing identity equivalence in multi-relational domains. In: Proceedings of the International Conference on Intelligence Analysis (IA-05). (2005)
9. Gil, Y., Ratnakar, V., Deelman, E., Mehta, G., Kim, J.: Wings for Pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In: The Nineteenth Innovative Applications of Artificial Intelligence Conference (IAAI-07). (2007)
10. Corkill, D.D.: Collaborating software: Blackboard and multi-agent systems and the future. In: Proceedings of the International Lisp Conference. (2003)
11. Klyne, G., Carroll, J.J.: Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation, W3C (February 2004) http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.
12. McGuinness, D.L., van Harmelen, F.: Owl web ontology language overview (2004) World Wide Web Consortium (W3C) Recommendation, at http://www.w3.org/TR/owl-features/.
13. Seaborne, A., Prud'hommeaux, E.: SPARQL query language for RDF. W3C recommendation, W3C (January 2008) http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.
14. Beckett, D.: RDF/xml syntax specification (revised). W3C recommendation, W3C (February 2004) http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.
15. Wolverton, M., Harrison, I., Martin, D.: Issues in algorithm characterization for link analysis. In: Papers from the AAAI Fall Symposium on Capturing and Using Patterns for Evidence Detection. (2006)
16. Sycara, K., Wido, S., Klusch, M., Lu, J.: LARKS: Dynamic matchmaking among heterogeneous software agents in cyberspace. Journal of Autonomous Agents and Multi-Agent Systems **5**(2) (June 2002) 173–203
17. Cheyer, A., Martin, D.: The Open Agent Architecture. Journal of Autonomous Agents and Multi-Agent Systems **4**(1) (March 2001) 143–148
18. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: Owl-s: Semantic markup for web services (2004) W3C Member Submission 22 November 2004, at http://www.w3.org/Submission/2004/07/.
19. Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., Tabet, S.: Semantic web services framework overview (2005) W3C Member Submission 22 November 2004, at http://www.w3.org/Submission/2004/07/.
20. Bruijn, J.D., Lausen, H., Polleres, A., Fensel, D.: The web service modeling language WSML: An overview. Technical Report 2005-06-16, DERI (2005) at http://www.wsmo.org/wsml/wsml-resources/deri-tr-2005-06-16.pdf.
21. Osterweil, L.J., Wisel, A., Clarke, L.A., Ellison, A.M., Hadley, J.L., Boose, E., Foster, D.R.: Process technology to facilitate the conduct of science. In: Unifying the Software Process Spectrum. Springer (2006) 403–415