An Agent-Based Interface to Terrestrial Ecological Forecasting

Keith Golden Ramakrishna Nemani Wanlin Pang† Petr Votava‡ Oren Etzioni
NASA Ames Research Center
MS 269-2
Moffett Field, CA 94035
{keith.golden | rama.nemani}@nasa.gov
University of Washington
Box 352350
Seattle, WA 98195-2350
etzioni@cs.washington.edu
† QSS Group, Inc.
‡ California State University, Monterey Bay

***Abstract*— This paper describes a flexible agent-based ecological forecasting system that combines multiple distributed data sources and models to provide near-real-time answers to questions about the state of the Earth system. We build on novel techniques in automated constraint-based planning and natural language interfaces to automatically generate data products based on descriptions of the desired data products.**

I. Introduction

The latest generation of NASA Earth Observing System (EOS) [1] satellites has brought a new dimension to continuous monitoring of the living part of the Earth System, the biosphere. EOS data can now provide weekly global measures of vegetation productivity and ocean chlorophyll, and many related biophysical factors such as land cover changes or snowmelt rates. However, the highest economic value would come from forecasting impending conditions of the biosphere, to allow decision makers to mitigate dangers or exploit positive trends. NASA's strategic plan for the Earth Science Enterprise identifies ecological forecasting as a focus for research. Ecological forecasting predicts the effects of changes in the physical, chemical and biological environment on ecosystem activity. Possible applications of such a system include predicting shortfalls or bumper crops of agricultural production, populations of threatened or invasive species or wildfire danger in time to allow improved preparation and logistical efficiency.

Petabytes of remote sensing data are now available to help measure, understand and forecast changes in the Earth system, but using these data effectively can be surprisingly hard. The volume and variety of data files and formats are daunting. Simple data management activities, such as locating and transferring files, changing file formats, gridding point data, and scaling and reprojecting gridded data, can consume far more personnel time and resources than the actual data analysis. Some scientists commit to a particular data source or resolution just because using anything different would be more effort that it's worth.

This project is supported by the NASA Earth Science REASoN program (Research, Education, and Applications Solutions Network). Early support was provided by the NASA CICT Intelligent Systems program.

Better tools can help, but most of the tools developed to date are little more than shell scripts; they lack the flexibility to meet the diverse needs of users and are difficult to extend to handle changes in available data sources.

We are developing a more adaptable solution, a ***soft***ware ro***bot***, or *softbot* [2] (also known as a software agent), a sophisticated computer program to which a person can delegate tasks. Our softbot, called IMAGEbot, is based on automated constraint-based planning and a flexible component-based architecture. Unlike script-based approaches, where the instruction sequences for managing and processing data are hand-coded, in our softbot-based approach, the instruction sequences are automatically generated based on user requests and available data sources. New data sources, models or data-processing programs can be added in a plug-and-play fashion, and the planner can adapt to errors or data dropouts by trying alternative ways of achieving the same goal, such as using other, possibly lesser quality, data sources.

We have demonstrated this technology in the Terrestrial Observation and Prediction System (TOPS), an ecological forecasting system that assimilates data from Earth-orbiting satellites and ground weather stations to model and forecast conditions on the surface, such as soil moisture, vegetation growth and plant stress. The planner identifies the appropriate input files and sequences of operations needed to satisfy a data request, executes those operations on a remote TOPS server, and displays the results, quickly and reliably.

A. Overview

The architecture of the IMAGEbot agent is described in Fig. 1. The major components of this architecture can be executed on different machines and communicate over the Internet, and the execution of plans can also be distributed, to exploit the intrinsic parallelism of dataflow plans. In the remainder of the paper, we describe a few of the components of this architecture in more detail:

- **DPADL:** Section II discusses the Data Processing Action Description Language (DPADL) [3], which is used to provide action descriptions of models, filters and other programs as well as descriptions of available data sources.

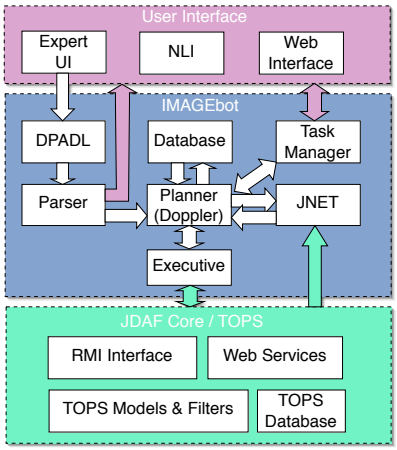

Fig. 1. The IMAGEbot agent architecture

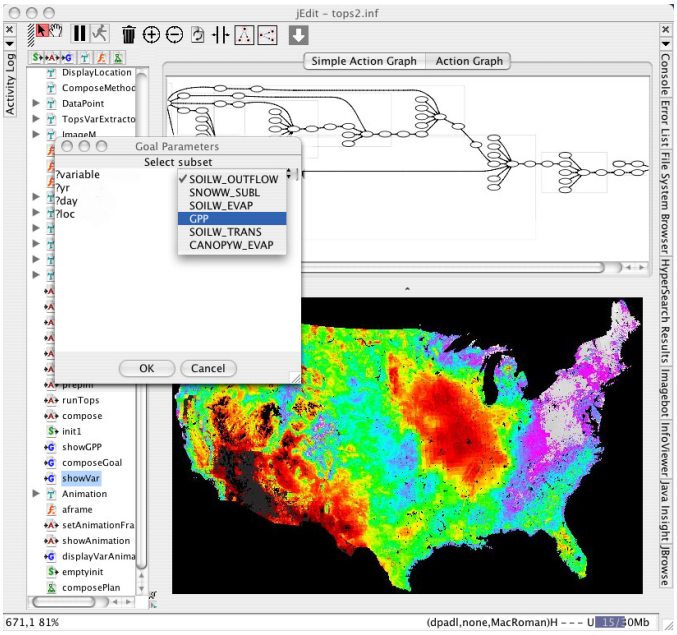

Fig. 2. The IMAGEbot expert UI provides advanced capabilities for editing and debugging models, filters and data sources (left panel and DPADL editor window, not shown), inspecting and modifying dataflow plans (top panel), and viewing the results of plan execution (bottom panel).

Goals, in the form of data product requests, can also be described in DPADL. DPADL is an expressive, declarative language with Java-like syntax, which allows for arbitrary constraints and embedded Java code.

- **User Interface:** Section III discusses the user interface. Although DPADL is a powerful, expressive language, it is more appropriate for programmers than naive users. We provide a simplified form-based Web interface to allow users to submit typical requests. For more advanced use, we are also developing a natural language interface (NLI), which will allow complex data requests to be posed in an intuitive manner.
- **DoPPLER Planner:** Section IV discusses the planner, which accepts goals in the form of data descriptions and synthesizes dataflow programs using the action descriptions read in by the DPADL parser, consistent with information stored in the database.
- **JNET:** Section V discusses the constraint solver, JNET, which can handle numeric and symbolic constraints, as well as constraints over strings and even arbitrary Java objects. The latter are evaluated by executing the code embedded in constraint definitions, specified in the DPADL input file.
- **JDAF:** Section VI describes JDAF, a framework that provides a common API for all TOPS data-processing programs and models for ecosystem forecasting.

B. Ecological Forecasting

As a demonstration of our approach, we have applied IMAGEbot to the Terrestrial Observation and Prediction System (TOPS, http://www.forestry.umt.edu/ntsg/Projects/TOPS/), an ecological forecasting system that assimilates data from Earth-orbiting satellites and ground weather stations to model and forecast conditions on the surface, such as soil moisture, vegetation growth and plant stress [4]. Prospective customers of TOPS include scientists, farmers and land managers. With such a variety of customers and data sources, there is a strong need for a flexible mechanism for producing the desired data products for the customers, taking into account the information needs of the customer, data availability, deadlines, resource usage (some models take many hours to execute) and constraints based on context (a scientist with a palmtop computer in the field has different display requirements than when sitting at a desk). IMAGEbot provides such a mechanism, accepting goals in the form of descriptions of the desired data products.

The goal of TOPS is to monitor and predict changes in key environmental variables. Early warnings of potential changes in these variables, such as soil moisture, snow pack, primary production and stream flow, could enhance our ability to make better socio-economic decisions relating to natural resource management and food production [5]. The accuracy of such warnings depends on how well the past, present and future conditions of the ecosystem are characterized.

The inputs needed by TOPS include:

- Fractional Photosynthetically Active Radiation (FPAR) and Leaf Area Index (LAI)
- Temperatures (minimum, maximum and daylight average)
- Precipitation
- Solar Radiation
- Humidity

We have several potential candidate data sources at the beginning of each model run. The basic properties of the inputs are listed in Table I. The specific data inputs that are selected will depend on goal constraints, such as requirements on resolution or coverage or resource limits.

In addition to the attributes listed in the table, data sources also vary in terms of quality and availability — some inputs are not always available even though they should be. For example, both the Terra and Aqua satellites have experienced technical difficulties and data dropouts over periods ranging from a few

TABLE I
TOPS INPUT DATA CHOICES

Source	Variables	Frequency	Resolution	Coverage
Terra-MODIS	FPAR/LAI	1 day	1km, 500m, 250m	global
Aqua-MODIS	FPAR/LAI	1 day	1km, 500m, 250m	global
AVHRR	FPAR/LAI	10 day	1km	global
SeaWIFS	FPAR/LAI	1 day	1km x 4km	global
DAO	temp, precip, rad, humidity	1 day	1.25 deg x 1.0 deg	global
RUC2	temp, precip, rad, humidity	1 hour	40 km	USA
CPC	temp, precip	1 day	point data	USA
Snotel	temp, precip	1 day	point data	USA
GCIP	radiation	1 day	1/2 deg	continental
NEXRAD	precipitation	1 day	4 km	USA

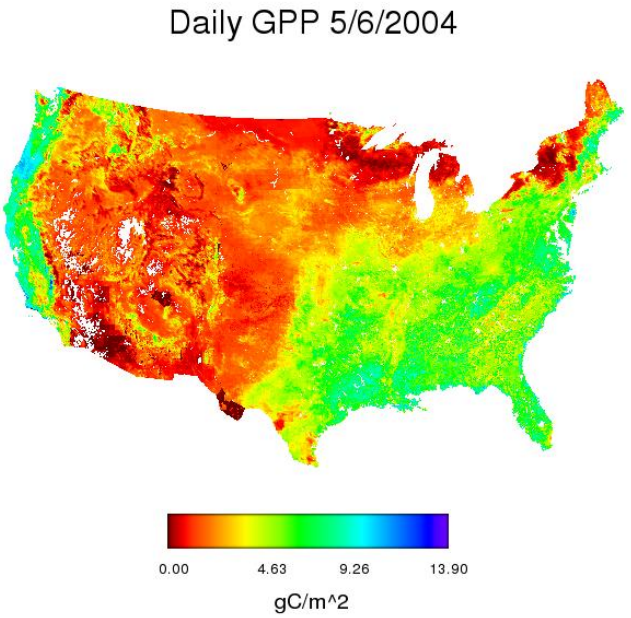

Fig. 3. The Terrestrial Observation and Prediction System (TOPS) generates daily nowcasts of biospheric variables, such as Gross Primary Production (GPP).

hours to several weeks. Depending on the data source, different processing steps are needed to get the data into a common format. We have to convert the point data (CPC and Snotel) to grid data, and we must reproject grid data into a common projection, subset the dataset from its original spatial extent and populate the input grid used by the model. The data are then run through the TOPS model, which generates desired outputs.

II. DPADL

The Data Processing Action Description Language (DPADL) [3] is a planning domain description language specialized for data processing domains. It differs from other domain description languages, such as PDDL [6] by describing characteristics typical of data processing domains, such as complex data structures (Fig. 4), object creation and copying, operations on large sets and integration of multiple software systems.

DPADL differs from metadata languages such as the Earth Science Markup Language (ESML) [17] in that DPADL is used to describe not only data, but the programs (models, filters, etc.) that process data, with a sufficient level of detail that it is possible to derive a DPADL description of the output of a dataflow plan given the DPADL descriptions of the inputs and the processing steps used to transform the inputs into the outputs. More importantly, the process can be reversed: given a DPADL description, it is possible to determine a set of data sources and a sequence of data processing steps that will produce a data product matching that description. That is the basis of how the DoPPLER planner works: Given a DPADL description of a requested data product (*i.e.*, a *goal*), the planner generates a sequence of data-processing steps (*i.e.*, a *plan*) that will produce data matching the description (*i.e.*, satisfying the goal).

Since it is used to describe goals as well as data and data-processing components, DPADL may be viewed as the "power user" interface to the planner, just as SQL provides a powerful interface to a database. DPADL is, in fact, far more expressive than SQL. However, just as most of us use databases on the Web every day without ever typing an SQL query, we don't expect typical users to formulate their data product requests using DPADL.

III. USER INTERFACE

We are developing two alternative ways of submitting data product requests. For naive users, we have developed a simple form-based interface that allows users to select a *goal template* and instantiate that template with specific parameters to customize the goal to their requirements. For example, we have a template to display a false-color image of a selected Earth-system variable. Parameters of the template include the specific variable to display, the geographic region and the date of interest. The goal template itself is simply a parameterized DPADL goal, and the GUI form to customize it is generated automatically from the parameters of the goal, but the user needn't know anything about that. Goal templates can be fairly general, so just a handful of them can cover most the data product requests that casual users are likely to submit. However, the space of possible goals that can be specified by this method is clearly limited, and this approach will not be adequate for expert users.

Since expert users are likely to be scientists rather than programmers, requiring them to submit their data product requests

using DPADL is not realistic. Instead, we are developing a natural language interface (NLI). Although it will still be less expressive than DPADL itself, the NLI will allow users to pose a much richer set of requests than any form-based GUI, but without requiring them to learn a new language to do so. Our NLI is based on Precise, a novel Natural Language Interface for Databases (NLIDB) [7].

Precise takes English questions and maps them to the corresponding database queries, enabling scientists who are not database programmers to formulate their queries in English. Precise combines lexical constraints, syntactic constraints from the English question, and semantic constraints from the database to rapidly narrow down the possible interpretations of a question. When multiple interpretations are possible, Precise asks the user to clarify the intended interpretation. Precise has two important properties that make it well suited for this project: portability and reliability. Precise is not tied to any particular database. Instead, it automatically generates its lexicon based on the vocabulary used in the database, and its semantic constraints are extracted from the schema of the database.

As a result, it is convenient to port Precise for use on a broad range of databases. Precise takes several steps to ensure that it is reliable. In contrast with other systems, Precise analyzes every word in the user's question. In addition, when it encounters ambiguity it refuses to settle for one of several interpretations. Instead, it asks the user to clarify the question in a manner that enables Precise to converge on the appropriate database query.

We are in the process of adapting Precise to the task of generating DPADL queries and we will develop simple dialog strategies that help guide users towards clarifying their information requests. This is essential to enable users familiar with the Earth science domain but not familiar with the technology to specify data requests with minimal training.

IV. DoPPLER Planner

Data processing has traditionally been automated by writing shell scripts. There are some situations when scripts are the best approach: namely, when the same procedure is to be applied repeatedly on different inputs, the environment is fairly stable and there are few choices to be made. However, in many applications, including TOPS, none of these assumptions holds. There are many different data products we would like the system to produce, there are many inputs and data-processing operations to choose from in producing those products, and the availability of these inputs can change over time. To address these challenges, we developed a planner, called DoPPLER (Data Processing PLannER), to automate data processing.

For our purposes, planning is a restricted form of automatic program synthesis, in which a plan, typically a loop-free sequence of actions, such as data-processing operations, is generated in response to a goal, a set of conditions that the plan must bring about. The goals DoPPLER accepts are descriptions of desired data products, and the plans it generates are dataflow programs, which produce the requested data.

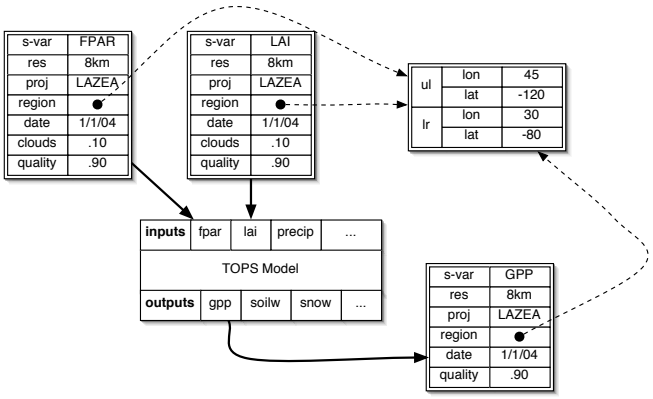

Fig. 4. Structured inputs and outputs to a TOPS model

A dataflow program is composed of data-processing operations, each of which can have multiple inputs and outputs, in which outputs of one action can be connected to inputs of another. Actions are eligible for execution as soon as their inputs are available, and multiple actions can be executed in parallel. This approach is well suited to supporting interoperability among even legacy systems, because the planner can be used to connect these systems together in whatever way needed to achieve a particular goal. All that is needed is descriptions, in the form of planner actions, of the systems to be integrated. The approach leads to a design that is modular and evolvable, since any new component can be brought in by providing only a description of that component. Similarly, descriptions of individual components can be modified, and descriptions can exist for different versions of components, leading to a system that is maintainable. Libraries of such descriptions, which we call domain libraries, can be distributed along with the components they describe, leading to a plug-and-play architecture.

There are significant differences between Earth Science data processing and more traditional planning domains, which calls for different techniques. Notable features of data processing domains include large dynamic universes, large plans, incomplete information and uncertainty.

A. Decisions, decisions

As we discussed in Section I-B, we have a number of inputs to choose from, which are applicable under different circumstances. The data may come from several satellites, ground stations, or as outputs from other models, forecasts and simulations.

In addition to input choices, we also have several choices of models to use with the data. As with the data, the models produce results of various quality, resolution, and geographic extent. Moreover, there may sometimes be significant trade-offs in performance versus precision. An FPAR/LAI algorithm provides a good example of this trade-off. We can produce an FPAR/LAI pixel using either a lookup table or a radiative transfer method [8]. In the case of a lookup table, we derive a Normalized Difference Vegetation Index (NDVI) from two surface reflectance channels by a means of a simple equation,

and then use the NDVI value together with its landcover value as a key into a static lookup table that will give us the FPAR and LAI values. The complexity of this algorithm is O(1). On the other hand, we can use the radiative transfer method, which contains a large number of intermediate computations and has complexity O($n \log n$). This fact, together with the number of runs we may attempt, translates into a substantial difference in user time, and while the radiative transfer method provides us with good results, it is not suitable for more interactive or first-pass applications, where the lookup table is sufficient. In these first-pass applications, we are looking for large abnormalities and deviations from long term normals, so high precision runs do not necessarily provide us with better results.

Another reason for using different models at different times is their possible regional character. Some models are highly specialized and provide very good and precise results in only certain parts of the world. This is partially due to the fact that the scientists who develop these models have a great deal of knowledge about specific geographic areas (Pacific Northwest, the Amazons, etc.). They have collected large amounts of local data over the years, and were able to develop models whose outputs are highly accurate in these regions. We usually don't want to use these models when we are concerned with global monitoring, but they are useful when we have identified an important event occurring at the region where we have a very accurate regional model.

B. Large dynamic universes

Over the last decade, great improvements have been made in the efficiency of planning and scheduling algorithms, thanks largely to the International Planning Competition and a corresponding set of benchmark planning problems that make such bake-offs possible. Unfortunately, this focus on benchmark problems has resulted in planners that are specialized for "puzzle problems," which are very complex but nonetheless quite small in terms of the number of objects that must be manipulated. Not only is the number of objects in these problems small, it is completely known and unchanging. Data processing domains are of a different character altogether. They are not as complicated as the benchmark "puzzle problems," but they can be much larger, with thousands or millions of objects (such as data files), which are generally impossible to identify in advance. Furthermore, most actions create new objects, so the universe is not even static. Inspection of the planning problems from the Third International Planning Competition (IPC3) reveals that even the hard problems typically have fewer than 100 objects total. In contrast, if we consider a single product from a single instrument (MODIS) on a single satellite (say, Terra) for a single day, there are 288 tiles. To produce a given data product, we may need to consider multiple products from multiple instruments, residing on multiple satellites, and multiple days' worth of data.

Despite these differences, we would still like to benefit from the progress that has been made in developing fast planning algorithms, so we have adapted one of the principle techniques for fast planning, *planning graph analysis* [9] to work in a representation in which the objects cannot all be explicitly

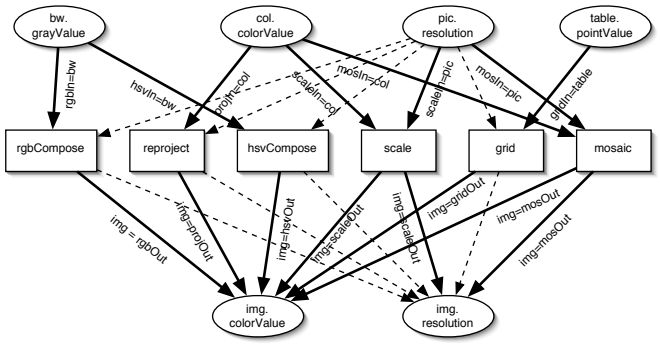

Fig. 5. Lifted planning graph with constraints

specified in advance. We have developed a representation, called a *lifted planning graph* (Fig. 5), in which variables are used to represent sets of possible objects. A planning graph (lifted or otherwise) is a layered graph representation of a planning problem in which the first layer consists of all conditions (represented in Fig. 5 using elliptical nodes) that are true at the start, the second layer consists of all actions (represented by rectangular nodes) whose preconditions are satisfied by the conditions in the first layer, the third layer consists of conditions that are enabled by the actions in the second layer, and so on. The graph continues in this way, with alternating condition and action layers, each layer providing support for the next. The first level that a condition appears at in the graph provides a lower bound on the number of steps needed to achieve it. This lower bound proves to be a quite useful distance estimate in heuristic search algorithms, such as A*.

In a traditional planning graph, each action node corresponds to a single ground action (*i.e.*, an action with all its parameters specified). For example, if the `scale` action takes a single file as input and a single numerical value specifying the scale factor, and if there are 5 input files and 5 possible scale factors, then there will be 25 instances of `scale`, one for each pair of input file and scale factor. Of course, in practice there will be many possible input files and infinitely many possible scale factors, so the traditional approach doesn't work. In our lifted planning graph, we represent both the input file and the scale factor using variables, so only one instance of `scale` is needed. Then we use *constraints* to specify how these variables depend on other variables in the planning graph. For example, the size of the output of `scale` is a product of the size of the input and the scale factor. If the size of the output is determined by constraints on the goal and the size of the input is determined by the set of candidate images, then then the set of possible scale factors can be determined. We have developed a novel constraint propagation algorithm to perform this sort of reasoning on a lifted planning graph.

V. Constraint reasoning

Constraints appear at all levels in data-processing domains.

- At the problem level, we have constraints on time and resource consumption. For example, one of the objectives of the TOPS system is to perform the complete processing

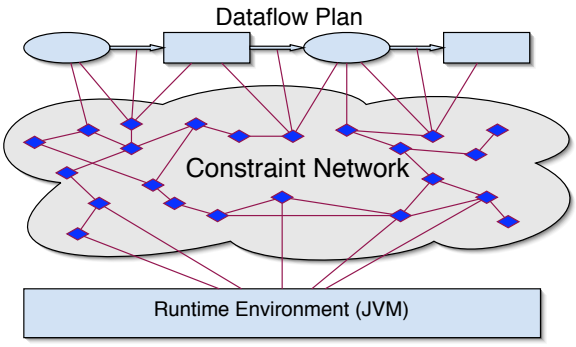

Fig. 6. Constraints in JNET can reference objects and procedure calls in the Java Virtual Machine (JVM), providing the "glue" between a dataflow plan and the runtime environment in which it is executed.

and analysis of data for a particular day no later than 8am the following day. If we have an algorithm that runs for 10 hours and we know that the last data for the current day will be arriving around midnight, we cannot accomplish the goal and we should consider another algorithm.

- At the file level, we can have constraints on size, quality, etc. For example, we may not want to process files for regions with more than 80% cloud cover. In this case, we may have to use a different, and less cloudy, source of data.
- At the pixel level, constraints may specify subsets of one or more datasets. For example, we may want to process data only for a certain country or region, or we may want to run an algorithm only during certain time periods. We may want to run the algorithm only on pixels of certain underlying type; for example, only for broad-leaf forests. Finally, during validation, we often compare satellite data with ground measurements, and we are only interested in specific points on the ground where we have validation measurements.

In order to deal with the many constraints that arise in a plan, we have developed a constraint reasoning system called JNET. As we discussed, JNET supports a novel algorithm for constraint propagation over lifted planning graphs. It also supports a number of other novel features, including:

- Powerful support for constraints on strings [10], useful for capturing the often complex file naming conventions as constraints between file pathnames and other file properties.
- Constraints over sets of objects [11].
- Constraints over complex structures, such as data structures (Fig. 4).
- Constraints over arbitrary Java objects, and defined in terms of arbitrary Java code. These are useful providing low-level integration between the planner and TOPS, which is written in Java (Fig. 6).

VI. Java Distributed Application Framework (JDAF)

In order to facilitate interoperation of the planner with the Earth science processing algorithms, as well as general extensibility and flexibility of the overall system, we have implemented the Java Distributed Application Framework (JDAF). Using this framework, we are able to easily integrate existing algorithms written in several different languages (C, C++, Fortran) into a complex application. While the algorithm integration is an important feature of the system, there is a provision for another integration, equally important — integration of the acquired data needed for the processing. There has been an enormous increase in the data volume and the number of data sources over the past several years, and while some data are being duplicated (for example we can obtain FPAR/LAI data from MODIS-Terra, MODIS-Aqua, AVHRR, or MISR), they usually come in variety of formats ranging from simple binary to HDF-EOS. The different data formats often bring another complexity into the system integration process, because the system will require new I/O modules that can read these new formats. With these facts in mind, we are building our framework in a way that accommodates both data and algorithm fusion, so that we can add new algorithms and new data streams seamlessly to the existing system while minimizing the integration efforts.

Since most of the Earth science algorithms are written in C or C++, we take advantage of Java Native Interface (JNI) facilities provided by the standard Java distribution. There is a single point of entry into and out of the native code, and we only use the Java interface for parameter passing between the processing algorithm and the rest of the system. This leads to a very simple design and a fast and efficient integration. On the Java side of the system, we provide a set of common APIs, which are implemented by each of the active objects (data preprocessing objects, processing algorithms, data analyzers). This makes it simple to form processing pipelines in a flexible manner, by either an application programmer, or by the planner. The simplicity of integration, flexibility, and fast deployment makes JDAF a good candidate for prototyping of new algorithm processing systems, competing with scripting languages such as Perl. Even though scripts are very suitable for fast prototypes, JDAF adds the flexibility and the distributed execution component not often available in common scripting languages.

VII. Related Work

There has been little work in planner-based automation of data processing. Two notable exceptions are Collage [12] and MVP [13]. Both of these planners were designed to provide assistance with data analysis tasks, in which a human was in the loop, directing the planner. In contrast, the data processing in TOPS must be entirely automated; there is simply too much data for human interaction to be practical. Pegasus [14] is a workflow planning system for computation grids, a problem similar problem to ours, though their focus is on mapping prespecified workflows onto a specific grid environment, whereas our focus is on generating the workflows.

Planning for data processing shares many characteristics with planning for information integration and planner-based software agents [15], [2]. The primary difference is the need in data-processing plans to reason about information that will never be known to the agent but is nonetheless essential to the

task at hand — namely, the information contained in the data files that the agent must process.

A number of frameworks are being developed for improving interoperability among different data systems, such as the Earth System Modeling Framework [16] and the Earth Science Markup Language [17]. What sets our work apart is the use of planning and scheduling software to automate the generation of data products based on user goals. Our aim is not to establish a competing standard but to support (and exploit) standards whenever they exist. For example, we are using ESML libraries in JDAF and we intend to support the translation between ESML metadata and DPADL data descriptions. However, expecting all systems to converge on a common standard is probably unrealistic. Our approach can help bridge the gap between legacy systems and emerging standards.

The EnVironmEnt for On-Board Processing (EVE) [18] is an execution framework for data-processing plans to be run on-board an Earth-orbiting satellite. Unlike IMAGEbot, EVE provides no planning capabilities; plans are generated by humans.

The Amphion system [19] was designed to construct programs consisting of calls to elements of a software library. Amphion is supported by a first-order theorem prover. The task of assembling a sequence of image processing commands is similar to the task Amphion was designed to solve. However, the underlying representation we use is a subset of first-order logic, enabling the use of less powerful reasoning systems. The planning problem we address is considerably easier than general program synthesis in that action descriptions are not expressive enough to describe arbitrary program elements, and the plans themselves do not contain arbitrary loops or conditionals.

REFERENCES

[1] M. King and R. Greenstone, "1999 EOS reference handbook, a guide to NASA's earth science enterprise and the earth observing system," 1999. http://eos.nasa.gov.

[2] O. Etzioni and D. Weld, "A softbot-based interface to the Internet," *C. ACM*, vol. 37, no. 7, pp. 72–6, 1994.

[3] K. Golden, "DPADL: An action language for data processing domains," in *Proceedings of the 3rd NASA Intl. Planning and Scheduling workshop*, pp. 28–33, 2002. to appear.

[4] R. Nemani, P. Votava, J. Roads, M. White, P. Thornton, and J. Coughlan, "Terrestrial observation and prediction system: Integration of satellite and surface weather observations with ecosystem models," in *Proceedings of the 2002 International Geoscience and Remote Sensing Symposium (IGARSS)*, 2002.

[5] R. Nemani, M. White, P. Votava, J. Glassy, J. Roads, and S. Running, "Biospheric forecast system for natural resource management," in *Proceedings of GIS/EM -4*, 2000.

[6] M. Fox and D. Long, "PDDL 2.1: An extension of PDDL for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, 2003.

[7] A.-M. Popescu, O. Etzioni, and H. Kautz, "Towards a theory of natural language interfaces to databases," in *IUI*, 2003.

[8] Y. Knyazikhin, J. Glassy, J. L. Privette, Y. Tian, A. Lotsch, Y. Zhang, Y. Wang, J. T. Morisette, P. Votava, R. B. Myneni, R. R. Nemani, and S. W. Running, "MODIS Leaf Area Index (LAI) and Fraction Of Photosynthetically Active Radiation Absorbed by Vegetation (FPAR) Product (MOD15) Algorithm Theoretical Basis Document," 1999. http://eospso.gsfc.nasa.gov/atbd/modistables.html.

[9] A. Blum and M. Furst, "Fast planning through planning graph analysis," *J. Artificial Intelligence*, vol. 90, no. 1–2, pp. 281–300, 1997.

[10] K. Golden and W. Pang, "Constraint reasoning over strings," in *Proceedings of the 9th International Conference on the Principles and Practices of Constraint Programming*, 2003.

[11] K. Golden and J. Frank, "Universal quantification in a constraint-based planner," in *Proc. 6th Intl. Conf. AI Planning Systems*, 2002.

[12] A. Lansky, "Localized planning with action-based constraints.," *Artificial Intelligence*, vol. 98, no. 1–2, pp. 49–136, 1998.

[13] S. Chien, F. Fisher, E. Lo, H. Mortensen, and R. Greeley, "Using artificial intelligence planning to automate science data analysis for large image database," in *Proc. 1997 Conference on Knowledge Discovery and Data Mining*, Aug. 1997.

[14] J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta, and K. Vahi, "The role of planning in grid computing," in *Proc. 13th Intl. Conf. on Automated Planning and Scheduling (ICAPS)*, 2003.

[15] K. Golden, "Leap before you look: Information gathering in the PUCCINI planner," in *Proc. 4th Intl. Conf. AI Planning Systems*, 1998.

[16] A. da Silva, C. DeLuca, V. Balaji, C. Hill, J. Anderson, B. Boville, N. Collins, T. Craig, C. Cruz, D. Flanigan, B. Hallberg, M. Iredell, R. Jacob, P. Jones, B. Kauffman, E. Kluzek, J. Larson, J. Michalakes, D. Neckels, W. Sawyer, E. Schwab, S. Smithline, Q. Stout, M. Suarez, A. Tayanov, S. Vasquez, J. Wolfe, W. Yang, M. Young, and L. Zaslavsky, "The Earth system modeling framework," in *3rd NASA Earth Science Technology Conference*, 2003.

[17] R. Ramachandran, S. Graves, H. Conover, and K. Moe, "Earth science markup language," *Journal of Computers and Geosciences*, 2003.

[18] S. Tanner, K. Keiser, H. Conover, D. Hardin, S. Graves, K. Regner, R. Wohlman, R. Ramachandran, and M. Smith, "EVE: An on-orbit data mining testbed," in *Proceedings of the IJCAI workshop on Knowledge Discovery from Distributed, Heterogeneous, Autonomous, Dynamic Data and Knowledge Sources*, Aug. 2001.

[19] M. Stickel, R. Waldinger, M. Lowry, T. Pressburger, and I. Underwood, "Deductive composition of astronomical software from subroutine libraries," in *Proceedings of the 12th Conference on Automated Deduction*, 1994.