A Workflow Model for Earth Observation Sensor Webs

Robert Morris
Jennifer Dungan
NASA Ames Research Center
Moffett Field, California
Email: robert.a.morris@nasa.gov, jennifer.l.dungan@nasa.gov

Petr Votava
California State University, Monterey Bay
NASA Ames Research Center
Moffett FIeld, CA
Email:petr.votava@nasa.gov

***Abstract*—An Earth science *sensor web* consists of a distributed collection of sensors, Earth science models, human scientists and information technologists, and data archives. The scientific use of the sensor web consists broadly of seeking to improve the understanding of natural processes occurring on the Earth's surface or in the atmosphere. Sensor measurements serve to quantify aspects of these processes that allow Earth science models to make predictions of scientific and social value. The management problem for sensor webs considered here is the problem of *reconfiguring* the sensor web in order to answer new science questions. The notion of reconfiguration is used broadly here to describe a set of actions for retargeting sensors, querying databases for image data, or executing functions for analyzing acquired data. This paper describes a workflow model and architecture for a workflow management system for reconfiguring sensor webs.**

***Index Terms*—Sensor Webs, Workflow Management System**

I. Introduction

An Earth science *sensor web* consists of a distributed collection of sensors, Earth science models, human scientists and information technologists, and data archives. Sensors in a sensor web will typically vary with respect to the type of observations they make and with respect to the platform on which they reside: the platform can be fixed or mobile, ground-based, airborne, space-borne, etc.

The scientific use of the sensor web consists broadly of seeking to improve the understanding of natural processes occurring on the Earth's surface or in the atmosphere. Sensor measurements serve to quantify aspects of these processes that allow Earth science models to make predictions of scientific and social value. Other measurements are intended to improve the cohesion or consistency of the data of the sensor web itself: for example, measurements from one sensor can be used to validate observations taken by another.

The management problem for sensor webs considered here is the problem of *reconfiguring* the sensor web in order to answer new science questions. We use the notion of reconfiguration broadly to describe a set of actions for retargeting sensors, querying databases for image data, or executing functions for analyzing acquired data. Because the resources of a web are distributed, reconfiguration is partly a coordination problem that happens continuously. Sometimes, sensor web reconfiguration occurs at measured intervals, e.g. daily, as during a field campaign [9]. In other contexts, the web is reconfigured on demand, for example, in response to a significant event.

Currently, reconfiguring a sensor web is primarily conducted by humans, with varying degrees of automation. Human expertise in Earth science sensor web management comes in three broad types: *scientific* expertise to formulate requirements for observation, *data management* expertise to collect the data and set up and execute processing tasks and *mission planning* expertise for setting up and scheduling the sensing platforms to take measurements.

Much of the reconfiguration tasks performed by humans is tedious and time-consuming, and could be more effectively performed by machines. Recent advances in software automation can be applied to improving sensor web management by

- Making the web more *goal-directed*. If reconfiguration is directly tailored to specific science goals, the data acquired will be of more scientific value.
- Promoting *abstraction*. Humans can formulate goals at a high level, and the automated system will automatically work out the details in how to accomplish them.
- Making the web more *robust*. Unexpected changes to the sensor web that hinder its ability to be reconfigured in a planned way can be diagnosed and an alternative configuration that satisfies the same goals can be generated automatically.

This paper describes an approach to sensor web reconfiguration using workflows. Section 2 provides an overview of an architecture for sensor web management. Section 3 describes the processes and models for constructing workflows. Section 4 describes a model and process for automatically executing workflows. Section 5 describes the current state and future enhancements of the system.

II. Workflow Management for Earth Observation Data

A *workflow* is a specification of the data and control flow used to acquire and process data to accomplish a science or disaster mitigation goal. Most definitions distinguish between *abstract* and *concrete* workflows, depending on whether the services to be used are included in the specification. A *service* is any remotely accessible capability for transforming a set of inputs into outputs. Services are invoked from a client on

remote hosts using a remote invocation methods and protocols such as SOAP or Java/RMI. Services provide a layer between applications that use resources like sensors or data archives and the resources themselves. A *workflow management system* (WMS) is a set of software tools for defining, managing and executing workflows on a set of distributed resources.

NASA faces unique challenges in building tools to support the understanding of Earth processes that requires a concept of workflow that is similar to, but also different from, the notions of computational workflow for science [2] or workflows used for business practices [3]. NASA's unique role in conducting science, as well as engineering and operating the sensing resources used for conducting the science, implies that a workflow concept for NASA must simultaneously address the knowledge aspect (what questions related to science or disaster mitigation need to be answered through the acquisition and processing of data); the engineering aspect (what resources should be assigned to address the question); and the operations aspect (how do the assigned resources need to be reconfigured to acquire the needed data).

Typically, a WMS is defined as having two parts: a build tool and a run-time tool. The build tool consists of the software components and representations required to define and build workflows. The run-time tool consists of the software components and representations required to execute workflows. Workflows are executed by invoking services that access sensor web resources. Therefore, the WMS requires an interface to the web service layer.

A. Architecture

Figure 1 illustrates the architecture for the WMS described in this paper. The capabilities and models identified in the figure are described in detail below. The WMS build tool consists of a *goal specification tool*, an *abstract workflow generator* and a *concrete workflow generator*. The WMS run-time tool consists of an *executive*, a *web manager* and a collection of *providers*. The process of generating and executing workflows can be visualized by reading the figure from top to bottom. Human users define *Earth observation goals* with the aid of a goal specification tool and a goal library. The goal is transformed, either automatically using an *abstract workflow generator* or with human assistance, into an *abstract workflow* along with a collection of parameters that constrain the goal. Next, a *concrete workflow generator* transforms the abstract workflow into an executable concrete workflow, consisting of a *state transition network* and an *action mapping table*. The concrete workflow is executed automatically by an *executive*, which continuously updates the *workflow state information*. A workflow is executed by submitting *web resource requests* to the *web managers*, which is comprised of sub-managers for acquiring, processing and storing data. The appropriate sub-manager relays each request to the appropriate *provider* that in turn manage the interactions with the remote services that process the request. Finally, the services are directly accessed via "plug-ins", service invocation software functions associated with interface specification protocols, such as Sensor Observation Service (Part of the Open Geospatial Consortium specifications) [1].

B. Terrestrial Observation and Prediction System

The Terrestrial Observation and Prediction System (TOPS). TOPS [5] is being used as a testbed for the design and testing of the ideas presented in this paper. TOPS is a data and modeling software system designed to seamlessly integrate data from satellite, aircraft, and ground sensors with weather/climate and application models to produce operational 'nowcasts' and forecasts of ecological conditions [5]. TOPS provides instances of data, processing and storage providers invoked by the web managers to execute workflow actions. The examples of Earth science goals and workflows described in this paper are based on scenarios for data acquisition and processing used in TOPS.

III. Building Workflows

We introduce a process-based language for workflows with a graphical front end for specifying high level goals that correspond to them. We distinguish between three levels of abstraction in the process of building workflows: the *goal* level, the *abstract workflow level* and the *concrete workflow level*. The latter distinction is common in the terminology of workflows, but the goal level is introduced as a "scientist's view" of workflows.

A. Goal Specification

In order to simplify the process of programming the sensor web, it should be possible to describe a data acquisition workflow in a simple, intuitive way. There are two kinds of workflow specification languages: script-based (GridAnt, BPEL4WS) and graph-based (Symphony, DAGman Tool). Script-based languages are typically expressive enough to represent complex workflows but are often hard to use for non-programmers. Graph-based languages are easy to use and intuitive, but sometimes it is hard to express complex workflows using them.

We introduce a graphical Earth science goal specification language to represent Earth science goals as a scientist would tend to represent them. The language is based on a simple taxonomy of goal types and enables the expression of parameters that characterize the constraints on a desired data product, as well as simple rules for composing goals out of others.

The goal language is implemented using Cmap Tools [7]. Concept maps are graphical representations of knowledge in the form of concepts, joined graphically by links to express propositions. Concept maps were developed in the 1970s to track the progress of learning scientific knowledge by elementary students. *Cmap Tools* is a concept mapping suite developed at Institute of Human and Machine Cognition (IHMC). The tool possesses features to enable real-time collaboration, sharing, reuse of concept maps. Concept maps are used here to define the parameters and constraints on workflows used to acquire and process Earth observation data.

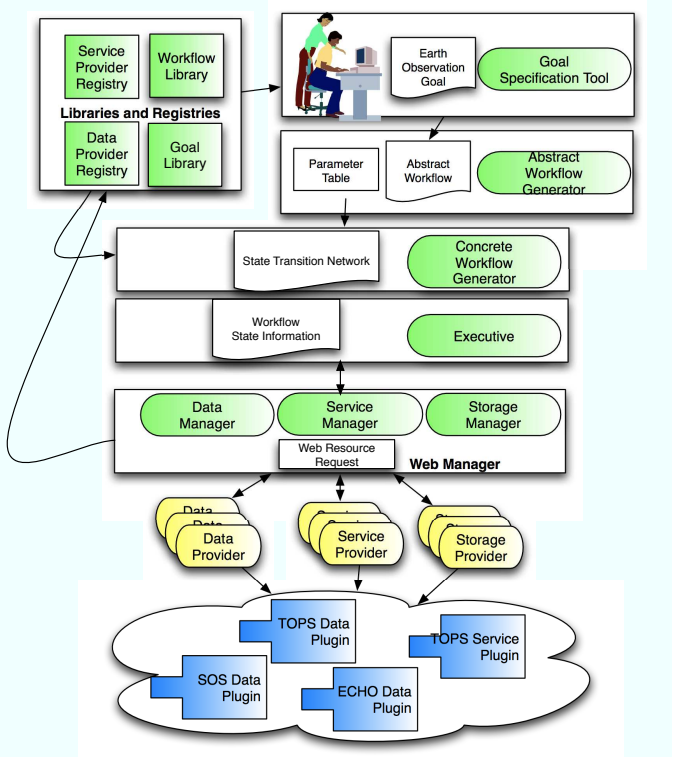

Fig. 1. Workflow Management System (WMS) architecture. The build tool consists of the goal specification tool, the abstract workflow generator, and the concrete workflow generator. The run-time tool consists of the executive, the web managers and the data, service and storage providers. The WMS is connected by the providers to a set of plug-ins to the service layer for accessing resources.

An Earth science *goal specification* defines an Earth science goal as a set of *property/value pairs* that characterize a desired data product. We use concept maps called Cmaps [6] to visualize a goal specification as a collection of labeled nodes and arcs.

The simplest goal type we refer to as *characterize*, which consists of four properties: an *attribute* that describes a type of data or measurement (the *what* of a goal), a *region* of interest (*where*), a *period* of interest (*when*), and a data *source*, which can be a sensor, a data archive, or a model (*how*). An example of a simple characterize goal is found in Figure 2. This Cmap specifies the goal: *characterize landcover over Sonora Mexico during 2001 using data source MOD12Q1*.

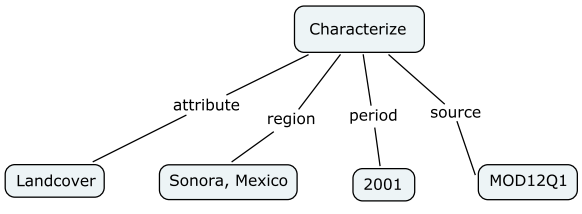

Fig. 2. Characterize Goal

Goals can be seen as composed out of other goals in two ways: through *elaborating* a value and through *conjoining subgoals*. An example of a goal conjoined from other goals is the goal to *compare* two data sources, e.g. output of a model or data set in terms of another data set. A special case of a compare goal is *validation* in which one data source becomes a reference to validate the accuracy of another.

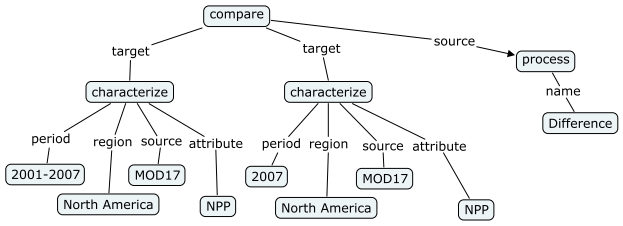

Fig. 3. Compare with Process Elaboration

In Figure 3, a comparison goal is expressed in which a long term average of Net Primary Production (NPP) data is compared against data from a single year. It is clear that a comparison goal arises from a composition of two characterize sub-goals. This example also illustrates a *process* elaboration, which is discussed further below. Another example of a composed goal is a *monitor* goal, an example of which is found in Figure 4. Such a goal can be seen as requiring a repeated characterization; the repeat pattern is specified through a *frequency* property. In this example, the frequency is dynamically determined by an elaboration called a *test*, discussed below.

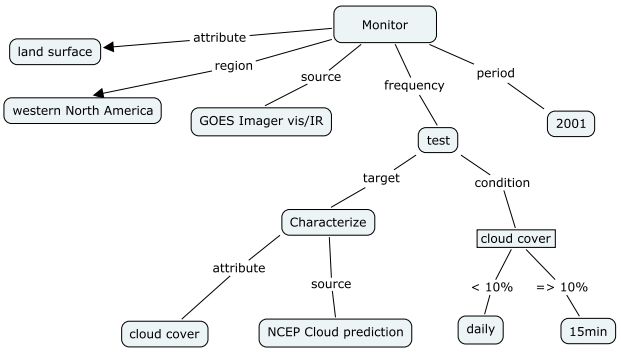

Fig. 4. Monitor Goal with A Characterize Sub-goal

There are two ways to elaborate a value: through a *test* and through defining a *process*. The need for a test elaboration may arise because the values of some properties (such as period, location, or source) are not known at specification time. The specification language allows users to assert that values are to be filled in at *workflow run time*. The way this is done is through asserting a *test* sub-goal. A test consists of a characterization of some property and a boolean condition. Depending on the truth of the condition, values of certain other properties, such as region, period or frequency, are assigned. An elaboration is depicted visually in Cmaps as an expansion of the depth of a goal tree.

An example of a test elaboration is found in Figure 4 in the context of a monitoring goal. A test consists of two attributes, a *target* and a *condition*. The target subgoal is a characterize goal. The attribute assigned in the characterize becomes a value that is tested by the condition. For example, if the attribute is *cloud cover*, as in the figure, the condition attribute must assert something about that attribute (in the example, whether some threshold value is exceeded).

The process elaboration allows for data to be processed as part of achieving a goal. A process elaboration always appears as a value of a *source* attribute (i.e., a process describes *how* some desired data product is to be attained). In Figure 3 a comparison is made of two data sets to determine the difference between the long-term NPP average over a given region and period with the average NPP for a single year. Computing the difference is performed by a function call. In general, a process invocation specification requires three fields: one or more *inputs* (data sources), one or more *parameters* that are required to run an algorithm.

Workflows tend to be highly structured objects that can be reused and revised. A *goal library* stores previously used goal specifications and templates for each of the goal types. In addition, at goal specification time, the user should be provided with information that allows for the values of goal attributes to be found. One way is through *registries* that define services and databases for acquiring the requested data. These registries could also be associated with access to *service discovery* engines.

In summary, a large class of Earth science investigations can be viewed as based on the goal of characterizing a phenomenon of interest, where to characterize a phenomenon means to define a set of values for time, space, measurement type and data or process source. From this foundation, other goals such as comparing data sets, or monitoring, can be composed. We defined two ways in which a goal can be elaborated: either by making data depend on the result of a test, or by invoking an algorithm for processing data as part of achieving the goal.

B. Workflow Generation

We have developed a system whereby a goal specification is automatically mapped into an executable workflow. To undertake the translation, the system uses a *process-based* model of workflows. Processes are executing sequences of actions, and a process language describes the possible states of the processes. We use a graphical representation called a *Labeled Transition System*[4] (LTS) for representing a workflow as a set of states and transitions of a finite state machine. Machine state transitions are modeled as labels consisting of names of actions, which eventually become instantiated as service invocations or other control actions. The building blocks of the workflow model, and their instantiations into abstract workflows, are described using an algebraic variant of the LTS notation called *Finite State Processes* (FSP). In FSP a process is a composition of simple expressions of the form $P : a \rightarrow Q$, which is interpreted as "To execute process P, perform action a and then execute process Q.

An abstract workflow defines the control logic and data flow without describing the services to be invoked to execute the workflow. This abstraction allows for reuse and sharing of workflows, and facilitates collaboration.

Actions are organized within workflows as concurrent processes. Broadly speaking, the workflow model has three parts: the *core*, where a set of atomic process definitions for simple sensor web service invocations and control actions are defined; a set of *workflow templates*, processes that correspond to workflow patterns found in high level goals such as monitor and validate; and a set of *composition rules* for instantiating workflows. Figure 5 illustrates the three components using FSP notation.

1) Core Workflow Model: There are two kinds of simple actions in a workflow process: *data acquisition* (DA) actions and *control* actions. DA actions directly map to sensor web services to acquire data; control actions allow for an execution system to initiate, monitor and control the state of workflow execution.

DA actions are the actions that combine to satisfy science goals. The model identifies three atomic DA actions: *acquire data*, *process data*, and *store data*, reflecting the major stages of requesting, processing, and retrieving data. The *acquire data* action requires a specification of a measurement, a location, a time, and the sensor used. The result of an acquire data action is a collection of measurements for the specified location and time. A *process data* action refers to any numerical or logical operation that extracts information of interest from

$$
\begin{aligned}
&\textit{Core Model} \\
GET &= (get \rightarrow END) \\
STORE &= (store \rightarrow END) \\
PROCESS &= (process \rightarrow END) \\
\\
&\textit{Workflow Templates} \\
ACQUISITION &= (ready \rightarrow get \rightarrow END \mid not\_ready \rightarrow ACQUISITION) \\
MONITOR\_TEST &= (done \rightarrow END \mid not\_done \rightarrow MONITOR) \\
CHARACTERIZE &= ACQUISITION; STORE; END. \\
MONITOR &= CHARACTERIZE; MONITOR\_TEST, \\
CHARACTERIZE\_TEST &= CHARACTERIZE; PROCESS; END. \\
\\
&\textit{Workflow Instantiation} \\
\|G1 &= (acq1 : MONITOR). \\
\|G2 &= (acq2 : CHARACTERIZE\_TEST). \\
\|C1 &= G2; G1; END. \\
\|CAMPAIGN\_ONCE &= (C1 \| G1 \| G2). \\
CAMPAIGN\_REPEAT &= CAMPAIGN\_ONCE; CAMPAIGN\_TEST, \\
CAMPAIGN\_TEST &= (campaign\_done \rightarrow END \mid campaign\_not\_done \rightarrow CAMPAIGN\_REPEAT). \\
\|CAMPAIGN &= (CAMPAIGN\_REPEAT).
\end{aligned}
$$

Fig. 5. Fragment of process-based workflow model, illustrating components and operations required to generate workflow instances. The *core model* defines simple processes for retrieving and processing data. *Workflow Templates* define recurring workflow patterns found in high level goals such as characterize and monitor; and a workflow *instantiation* using model components.

acquired data. Data process actions are typically repeated on different inputs. Finally, a *store data* action produces a data product required by the goals. Typically store actions consist of copying the output of a data processing action to a designated location for retrieval by the end user (the person issuing the initial request for the data).

Control actions can be used for three purposes. First, as a mechanism for *branching*, either as the result of choice or non-determinism. Choice control actions are associated with two or more transitions in an LTS, where only one of the triggers associated with the transition can be true. Branching as the result of non-determinism occurs as the result of the way the world is observed. For example, the result of requesting a sensor may or may not succeed, depending on things beyond the purview of the controlling system. Second, some control actions are used for *synchronization*, which is a way to impose an order on a set of DA actions. For example, a processing DA action may process data from an acquisition DA action. To enforce the dependency of the processing action on its inputs, a synchronizing action is introduced. Finally, a control action may be used to *wait* until some response from the sensor web is received. For example, a control action can be used to wait for notification that a set of measurements has been taken.

In Figure 5 (top) the core model is represented as four processes. The $ACQUISITION$ process consists of the control action *ready* that indicates when data are ready to be retrieved. It also contains a DA action *get* that fetches the data. The choice "|" operator allows for branching based on which control action $ready, not\_ready$ is enabled. The process also allows for repeating a sequence of actions (recursion) and thus allowing for waiting for data to be ready. This process model can be modified further to allow for aborting a process or other contingent action. The END process is a special process signifying completion of a process, and the "$\rightarrow$" symbol signifies the state transition as the result of performing an action.

Similarly, the $STORE$ and $PROCESS$ processes allow for constructing workflows that include actions for putting data somewhere to be retrieved by the requestor and to process image data.

2) Workflow Templates: The core workflow model defines simple processes out of atomic sensor web data acquisition and control actions. Workflow templates are representations of processes composed from other processes for the purpose of representing repeating workflow patterns, including those that correspond to high level goals. A library of workflow templates can be constructed and expanded over time, comprised of recurring patterns of workflow processes

The use of templates is illustrated in the middle section of Figure 5. First, a process called $ACQUISITION$

is defined to elaborate a get action with control actions $ready$ and $not\ ready$ to allow the execution system to wait until some data are ready to be acquired. Similarly, a $MONITOR\ TEST$ process is defined that allows data to be collected repeatedly over some period of time, again based on a control action $done$. A $CHARACTERIZE$ process is defined as a sequence of getting, processing and storing data for retrieval by the requestor (using ";" to indicate process sequencing). Similarly, a $MONITOR$ process is represented as a sequence of repeated $CHARACTERIZE$ processes, with a test to determine whether some threshold or other monitoring goal has been reached. Finally, the $CHARACTERIZE\ TEST$ is a process that allows for a process corresponding to a characterize goal that includes an elaboration sub-goal.

A library of workflow templates can be grown to represent patterns of workflow. Workflow templates are reusable scripts that make it easier to instantiate workflows. They are meant to be redundant in the sense that the set of legal workflow instances defined by the core model and composition rules is the same as the set generated with the addition of the templates. Nonetheless, they allow for more simplicity in running the model.

3) Workflow Instantiation: An abstract workflow is instantiated into a *concrete workflow* by a process of mapping actions or processes to specific services or data sources. A concrete workflow is an executable procedure, to be executed by an *executive*, described below.

Workflow instantiation is the process of generating an instance of an abstract workflow from user specifications. Workflow instantiation is performed automatically out of the following set of composition rules:

- *process instantiation*: defining a process as being an instance of a certain workflow template;
- *parallelization*: defining two or more processes as executing in parallel;
- *action* ($\rightarrow$) and *process (;) sequencing*: defining processes or actions as executing in sequence;
- *process recursion*: defining a process as a repeating execution of other processes; and
- *branching*: defining conditional transitions of processes.

Figure 5 (bottom) gives an example of generating a workflow instance using each of these rules. $acq1 : MONITOR$ defines $acq1$ as an instance of a monitoring process. $\|C1 = G2; G1; END$ defines process $C1$ as a sequence of other processes. $\|CAMPAIGN\_ONCE = (C1\|G1\|G2)$ defines a parallel process. Recursion and branching are defined in the same way that was illustrated in the template examples.

In summary, the workflow model for data acquisition on sensor webs can be summarized as follows:

- Basic actions come in two types: data acquisition (DA) actions and control actions;
- Workflows consist of processes composed out of operations for sequencing, repeating, choice, non-determinism and parallelism.
- A workflow model consists of three parts: the core model for defining basic actions as processes; a collection of workflow templates, which are scripts for defining workflow patterns; and workflow instantiations, which are automatically generated using composition rules.

C. Concrete Workflow Model

A concrete workflow is a binding of workflow actions to specific resources for execution. In the literature, a *workflow planning scheme* [8] describes the method for mapping abstract workflows into concrete workflows. Schemes are either static or dynamic depending on when bindings of tasks to resources occurs: static schemes perform all the bindings before execution, based on current knowledge about the available resources, whereas dynamic bindings can occur during execution. In this framework, we assume a static binding that is generated through user inputs. Workflow planning schemes can also involve criteria for optimization of execution performance. Again, the framework described here does not currently address these issues.

Binding DA actions involve introducing lower-level tasks for transferring data or requests for data. It requires a concrete workflow model that defines how tasks are to be executed on specific resources. Similarly, control actions are mapped into tasks for communicating with remote services, monitor the progress of a data acquisition task, or test the outcome of an analysis.

To facilitate the automatic execution of workflows, the concrete workflow is represented as a finite state machine. This representation is equivalent to the FSP format, but better enables execution a finite state processor, described below. The translation of a workflow in FSP format into a finite state representation is performed by the LTSA tool [4]. An example of a finite state representation of a concrete workflow is the following:

```
Q0 =
(not_ready_acq1 -> Q0 | ready_acq1 -> Q1),
Q1 = (get_acq1 -> Q2),
Q2 = (end_1 -> Q3),
Q3 = (process_p1 -> Q4),
Q4 = (end_2 -> Q5),
Q5 = (store_s1 -> Q6),
Q6 = (not_done_0 -> Q0  |done_0 -> Q7),
Q7 = STOP.
```

The Qi designate states, and the sequences identified with the states indicate the transitions, caused by invoking the actions. Each action, either a control or DA action, maps to an entry in an *action mapping table*.

This section has described a model for the automatic composition of workflows from high level specifications based on user goals. The following section describes a system for automatically executing workflows.

IV. Workflow Execution

The function of a workflow execution system (run-time tool) includes coordinating the remote resources used to carry out

data acquisition tasks, acquiring information from resources required for access, monitoring the execution of individual tasks and responding to execution failure. Execution systems are constrained to meet the requirements of the designers of the workflow being executed.

The system described here utilizes a *distributed hierarchical* approach to workflow execution (see Figure 1), with three levels: an *executive* being supported by a number of *web managers* that execute a portion of the workflow. Each lower level manager, in turn, has access to a number of *service providers* manage the execution of tasks on specific resources.

A. Executive

The executive is a 'high level' manager of workflow execution. The executive has complete access to the workflow state information, which summarizes the current state of workflow execution. It is the central coordinator of workflow execution, as well as a high level interface to the user and other elements of the workflow build system.

The executive is given as input the finite state representation of a concrete workflow and the associated parameter table. It continuously checks what actions are *enabled* (i.e. can be executed) and executes them. A control action is executed by an update to the state information to reflect the progress of the workflow. A DA action is executed by sending the information about the action to the appropriate web manager (discussed further below). After submitting the action, the executive will monitor the progress of the action and check for violations or anomalies during execution. Ultimately, the executive will need to initiate responses to failures in execution (e.g. in case a resource is not available). This capability of the executive is beyond the current scope of this project.

B. Web Managers

The web managers provide an added layer of distributed coordination between the executive and the web service layer. There are three kinds of web managers: a data manager, a process manager and a storage manager, which align with the three DA actions (acquire, process, store) at the workflow level.

The web managers are used in two phases of the overall data acquisition process: during the construction of a goal, and during execution. In the first phase, the web managers are used to facilitate *service discovery*. During goal creation the web managers assist the user in finding and selecting services or data sources to accomplish their goals. In the second phase, the web managers execute the desired data acquisition action using the providers.

Communication among the web managers, processes and datasets is primarily through the use of URN (Universal Resource Name). This nomenclature is chosen because of its compatibility with standards and protocols, such as OGC services, commonly in use. URNs uniquely identify the services and once URN is passed to the provider it is resolved, the particular process is loaded and executed. This provides a clean unified interface that only accepts URN and list of inputs and parameters.

The managers employ a flexible approach using plug-in services. Each manages a set of dynamically loaded service providers that provide access to various data resources, processes, and storage resources. For testing purposes with TOPS a *Local Service Provider* has been developed, which uses a SensorML [1] process model to describe local processes and the constraints on their inputs and outputs. However, we also plan to include the ability to access remote services (for example some of the functionality of TOPS can be accessed using web services).

C. Web Service Providers

The *web service providers* are loaded on demand from external configuration files or databases, and can be updated and reloaded during execution. Each represents a concrete provider on the Web, whether for data, such as a NASA DAAC, or processing capabilities. A provider also describes a set of available data services (e.g. SOS, ftp).

The services are implemented as a set of re-usable Java Plugin Framework (JPF) plug-ins that each deal with different communication protocols. Plug-ins have been widely used as a modular mechanism to extend the capabilities of host applications in many areas, including remote sensing applications (e.g. Google Earth). Plug-ins can be reused either with minimal customization for (SOS protocol plug-in) or more customization (file naming conventions for FTP protocol plug-in). The plug-in design enables customization to happen at configuration rather than source code level.

In order to provide the necessary support for the managers, each service plug-in must implement methods for service discovery, retrieval and status check. However, any service initialization and protocol details are handled internally by the plug-in without the data manager knowing about it. We are currently using a simple custom plug-in framework. Each provider manages a pool of threads, allowing for multiple requests to be handled simultaneously, thus supporting the concurrency inherent in the workflow model. The providers could also be run as a web service that handle multiple requests from number of different clients, not just the Web Managers.

V. Discussion

A prototype of the workflow management system just described is being tested using TOPS. Users of TOPS create goals using Cmaps. Goals currently supported include simple goals such as characterize as well as composite goals such as comparisons. A library of goal templates and previously used goal instances is maintained.

The Cmap tool generates translations of concept maps into an xml representation. The xml is annotated by our prototype with parameter information, as well as with URNs for accessing the specific data and processing services selected by the users. The workflow generator is then invoked to instantiate a workflow based on the goal inputs, as well as workflow

templates and libraries. The workflow is executed on TOPS automatically using the executive and managers.

The level of automation currently supported by the system is limited. More automation is required for it to be usable by a wider user group, including scientists. The set of required enhancements include:

- Extensions to the workflow build tool:
  - A user interface for machine-guided assistance for formulating Earth science goals, including automation for supporting goal editing, data and service discovery, and maintaining and sharing goal libraries;
  - Performance models for assisting the user in optimizing the selection of goals based on criteria such as cost and expected turnaround time for using a specific service;
  - A more powerful workflow language to support a wider class of workflows. Of immediate interest is the ability to represent workflows containing conditional behavior; for example, to execute a task only if some condition is true; and
  - A workflow library maintenance system for reusing workflow templates and instances.
- Extensions to the workflow execution tool:
  - Capabilities to support simultaneous monitoring of multiple workflow execution by the web managers;
  - Extensions to the executive and managers to support dynamic replanning of workflows as the result of task failures.

In addition to the above, we intend to further modularize the architecture to enable execution of the workflows on other execution systems such as the Business Process Execution Language (Bpel).

VI. Summary

This paper has introduced the design and implementation of a workflow management system for managing the acquisition, processing, and storing of Earth science data, the major components of what was referred to here as *sensor web reconfiguration*. The cornerstones of this system are

- a goal specification language that allows Earth scientists and other end users to construct high level goals for data acquisition,
- a process-based language of workflows for data acquisition that allows for the automated control of workflow execution,
- a hierarchical, distributed automated workflow execution system based on finite state control and
- a flexible interface to a web service layer that allows for added transparency in the types of service protocols utilized.

References

[1] M. Botts. SensorML: Standard for in-situ and remote sensors. In *Proceedings of EOGEO-2004*, June 2004. College London, London, UK.

[2] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *IEEE Computer*, 40(12):24–32, 2007.

[3] M. Jackson and G. Twaddle. *Business Process Implementation: Building Workflow Systems*. ACM Press/Addison-Wesley Publishing Co, 1997.

[4] J. Magee and J. Kramer. *Concurrency: State Models and Java Programming*. Wiley, 2006.

[5] 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 4th International Conference on Integrating GIS and Environmental Modeling (GIS/EM4)*, 2000. Banff, Canada.

[6] J. Novak. Concept mapping: A strategy for organizing knowledge. In S. Glynn and R. Duit, editors, *Learning Science in the Schools: Research Reforming Practice*, pages 229–245. Lawrence Erlbaum Associates, 1995.

[7] J. D. Novak and A. J. Caas. The theory underlying concept maps and how to construct them. In *Technical Report IHMC CmapTools 2006-01 Rev 01-2008*.

[8] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3:171–200, 2005.

[9] INTEX-B. http://www.espo.nasa.gov/intex-b/.