Deductive Synthesis of Workflows for e-Science

Bin Yang, Alan Bundy, Alan Smaill, Lucas Dixon *
School of Informatics, The University of Edinburgh
Appleton Tower, Crichton Street, Edinburgh EH8 9LE, UK
b.yang@ed.ac.uk, {bundy, smaill, ldixon}@inf.ed.ac.uk

Abstract

In this paper we show that the automated reasoning technique of deductive synthesis can be applied to address the problem of machine-assisted composition of e-Science workflows according to users' specifications. We encode formal specifications of e-Science data, services and workflows, constructed from their descriptions, in the generic theorem prover Isabelle. Workflows meeting this specification are then synthesised as a side-effect of proving that these specifications can be met.

1 Introduction

Data-intensive scientific computing usually involves very complicated processes that are normally composed of many steps, each involving significant computation. e-Science provides a distributed problem solving environment for many scientific projects and researchers, in which Grid computing is the key enabling infrastructure. Grid services are composed into workflows to build composite e-Science applications to achieve complex user tasks. Recent implementations of Grid computing environments [6, 16, 17, 18] provide workflow interfaces for users to construct composite Grid applications more efficiently, by either coding in particular workflow languages, manual composition in the provided GUI, or re-using expert-built templates.

Workflow-level access to Grid resources is a substantial step toward the realisation of the vision of e-Science. However, the e-Science Grid will keep expanding as more resources are incorporated into the e-Science community; the scale and complexity of such distributed and heterogeneous systems would make any level of assistance desirable to even experienced users. Several projects have addressed the problem and provided different levels of automation in helping users compose, validate and execute their workflows, such as [1, 11].

Our project focuses on the machine-assisted synthesis of workflows to meet users' specifications. In this paper, we describe a novel approach of modelling Grid data and services, in which they are extracted from their semantic descriptions in a formal system that is a further abstraction of the Virtual Data Language (VDL) [1]. We also demonstrate how we implement the formal system in the generic theorem prover Isabelle [14] to verify and synthesise abstract workflows that can be later translated for middleware through their workflow interfaces. Currently, our system can automatically verify that workflows meet their specification, but synthesising workflows from specification requires interactive guidance. We are working towards the automation of synthesis too.

2 Our Approach to Workflow Synthesis

Current workflow environments are still far from the reach of real scientists who are working with real scientific problems. To use a Grid through its workflow interface, many users have to struggle with the following obstacles:

- **Considerable Expertise** is required for scientists to effectively design their workflows using interfaces to script-based language [16, 18].
- **Service Look-up** is difficult for users as there are many service instances published by different service providers, intended for different disciplines, implemented over different platforms and with different levels of encapsulation.
- **Resource Deployment** is a prohibitive task for users due to its complexity. The abstract workflow has to be deployed onto a network of concrete Grid resources connected and configured accordingly.

In addition to the problems described above, there are two further difficulties that often arise in the manual composition approaches:

*This work was funded by EPSRC grant GR/S25388. We would also like to thank Ewen Maclean, Carole Goble, Jim Blyth and John Darlington for their helpful comments.

- Service instances have **Contextual Properties** which may need to be considered. For example, it may be preferable to incorporate services from a certain set of affiliated providers, or produce the computing result in a particular format depending on the local computing architecture. However, users are usually provided with limited information about the contextual properties of service instances. This is mainly due to the inadequate expressiveness of service description languages and the heterogeneity of the Grid environment and e-Science application domains.

- The complexity involved in the manual composition of workflows makes them likely to fail. The **Execution Failure** of composed workflow can be very hard to detect and avoid manually, as even advanced users cannot easily deal with all the issues stated above.

To address these problems, first, we adopt a semantic description of Grid resources. In our approach, Grid services, data and other Grid resources are declared as objects formally specified with their properties. Extra application-specific replicas of Grid services and data can be implemented for different computing architectures or stored in different locations, and they will have different *physical* properties, location, local name, run-time parameter, etc. However, the replicas remain *logically* equivalent in term of the metadata specifying their logical properties: ontological types and so on. We reason at this abstract logical level. Abstract workflows then need to be compiled into concrete ones, augmenting them with the details of actual Grid services. The compilation process is addressed by other projects [1], and we have put hooks into our representation to support its solution in the future.

Logic descriptions of Grid resources can be obtained through the *Metadata Catalog Service* (MCS) that responds to queries of domain-specific metadata and returns the logical names of matching Grid services or data [5, 15]. The physical locations and names can be retrieved by the *Replica Location Service* (RLS) [2] and *Monitoring and Discovery Service* (MDS) [3].

Second, Grid services and data are modelled as logical formulae. Our formalisation in Isabelle is rather straightforward, as all the property fields of Grid resource declarations are mapped into the Isabelle record datatype. We extract the ontology of data from the metadata used in existing Grid applications. For example, in the application domain of the Laser Interferonmeter Gravitational Wave Observatory (LIGO) [4], the data files recording a long sequential output of gravitational wave strain channels observed by LIGO instruments have the ontological type of `LIGO−pulsar`. There are other logical properties of `LIGO−pulsar` data, for example, the start, end positions, and the sample rate of the observation, as well as physical properties of concrete replicas, such as the URL to the server where the data file is stored and the local file name. This physical property information is stored in order to support subsequent compilation into a concrete workflow, even though we do not address the compilation process in our project.

In the data-centric e-Science environment, Grid services can be modelled as transformations on data. Therefore, the service metadata can be simply designed to include only the input and output data of the Grid service. Again, physical properties are used to declare the concrete details, including the location, the UNIX script and the arguments to access or invoke the service instance.

By allowing the declaration of pre-conditions, which can be the evaluations of the properties of the service and its in/output data, as well as the relationships between them. This allows the representation of knowledge about the contextual properties of Grid services, for example, how the service interacts with others, and other service-specific characteristics, requirements and capabilities.

With the application-specific knowledge of Grid services encapsulated in their independent specifications, we implemented generic workflow operators that depict the data-flow nature of e-Science Grid applications and are no longer specific to particular e-Science knowledge domains or applications.

The Isabelle theorem prover is used to prove that the specifications of the abstract workflow can be met, and the synthesised workflow is extracted from the proved theorem, *cf.* [12]. We use Isabelle to apply backwards search with unification to synthesise the structure of the workflow and lookup matching services. The workflow is specified by the accumulated instantiations of meta variables in the workflow. These instantiations bridge the gap between the source data and the desired data product. The proof both synthesises the workflow and verifies that this specification is met.

3 Modelling Grid Services and Data

Scientific experiments are data-centric: data are collected by measurement instruments, stored, processed and analysed; later it may be retrieved, compared and derived; or used by other researchers. e-Science is therefore data-centric as well. It is crucial to model the data objects, as well as the processes that transforms the data.

In a data-centric Grid, services can be modelled by the application of computational procedures that derive and transform data. Explicit representation of these procedures, so-called "virtual data", enables the recording of data provenance, discovery of matching services and on-demand data generation.

3.1 Semantical Description of Data

An extracted specification from an example Grid data file annotated with domain-specific knowledge from the LIGO application[1] [4] is shown below:

```
{ LIGO-pulsar;
    714265040 ;   low boundary
    714265294 ;   high boundary
    LSC-AS_Q H1 ILWD
    50.5 ; fcenter
    0.004 ; fband
    3 ; fderv1
    3 ;fderv2
    3 ; fderv3
    3 ; fderv4
    3 ; fderv5
    3 ; right ascension
    3 ; declination
    4096 ; sample rate  }
```

where $\texttt{LIGO}-\texttt{Pulsar}$ is the ontological type and other fields describe the logical properties of the abstract data stored in the data file. The ontology is extracted from the LIGO domain knowledge.

We now define the abstract logical representation of a data replica that store the pulsar data in the LIGO knowledge domain:

$$\forall\, pdata :: \texttt{PulsarData}.\ OntType(pdata) = \texttt{LIGO}-\texttt{Pulsar} \tag{1}$$

The generic type of Pulsar data is defined as an extensible record type [13] consisting of the following property fields:

$$\begin{aligned} \texttt{PulsarData} \equiv \{\, OntType &:: \texttt{Ontology}, \\ low &:: \texttt{Integer}, \\ high &:: \texttt{Integer}, \\ args &:: \texttt{String}, \\ &\vdots \\ sample{-}rate &:: \texttt{Integer}\,\} \end{aligned} \tag{2}$$

$OntType(\,)$ in (1) is a *Selector* function [13] that returns the value of the corresponding field of the data, which is "$\texttt{LIGO}-\texttt{Pulsar}$" in this case.

As shown above, we model the ontological knowledge of data using a record field of the more general record type, instead of a type in Type Theory. Our purpose in regarding the ontological type of data as one of the semantic properties is to (1) provide flexible access to the data ontology; (2) bridge the gap between the logical data discovery, by unification over the ontological type, and the instantiation of logical data to data instances, that can be regarded as extending the logical data with the instance-specific properties.

Unification is provided by Isabelle's implementation of Huet's higher order algorithm [8], and supports type as well as term variable instantiation. Unification is used to select applicable data and services which are represented using Isabelle's extensible record datatype [13]. We define the base record type `BaseData` that contains only the $OntType$ field, and then introduce sub-types (sub-classes) of Grid data, such as `PulsarData`, by extending the base type `BaseData` (super-class) with extra fields.

This representation allows unification to supports users querying e-Science data using domain knowledge and semantic descriptions expressed, such as:

$$\begin{aligned} \exists\, pdata :: \texttt{BaseData}-\texttt{Scheme}. \\ (OntType(pdata) = \texttt{LIGO}-\texttt{Pulsar}) \\ \wedge\, (low(pdata) = \texttt{1}) \\ \wedge\, (high(pdata) = \texttt{999}) \end{aligned} \tag{3}$$

The query can be read as: "Is there a data replica that is of the ontological type $\texttt{LIGO}-\texttt{Pulsar}$ and describes the LIGO sequence of the range $[1,\, 999]$?". Note that $pdata$ is existentially quantified. The proof of (3) will instantiate $pdata$. This instantiation will represent the desired abstract workflow. In this way, the proof synthesises the workflow.

$pdata$ can be instantiated to any data instance whose $OntType$ field is $\texttt{LIGO}-\texttt{Pulsar}$, as the data instances are partially evaluated solely by the $OntType$ field. The data instances are either manually declared or provided by RLS, and then translated to logical formulae as in (2). If there is such a logical representation of a data instance of $\texttt{LIGO}-\texttt{Pulsar}$ that stores the LIGO sequence of the first 1000 samples, then we are able to prove the formula (3) directly and thus $pdata$ is synthesised to be that data instance; otherwise, search is needed to find a combination of data instances that together satisfy the conditions in the conjunction. We employ backward search using unification with heuristics and tactics to guiding the proofs automation.

3.2 Specifying Grid Service with Data

To implement our more flexible and generic workflow synthesis system, we started by extending VDL to support types of data. The simple DAG shown in Figure 1 is an abstract LIGO workflow example. Below, we discuss how our system can synthesise this workflow from its specification.

We need to declare the new LIGO logical data and services that are specified as transformations of data:

$$rdata1,\, rdata2 \quad :: \quad \texttt{RawData}$$

[1] This uses the framework of Chimera and Pegasus. Thanks to Jim Blyth for providing the LIGO application example.

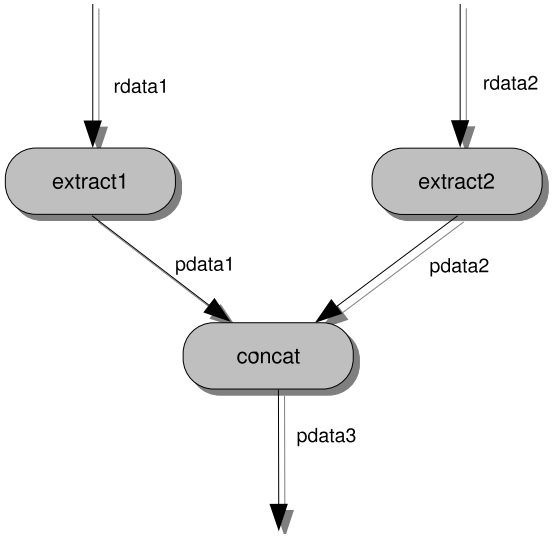

Figure 1. A Simple LIGO Workflow

$$pdata1,\ pdata2,\ pdata3 \ :: \ \texttt{PulsarData}$$

$$extract1,\ extract2 \ :: \ \texttt{Extract}$$
$$concat \ :: \ \texttt{Concat}$$

where the data type `RawData` is defined in a way similar to `PulsarData`; the $OntType$ fields of $rdata1$, $rdata2$ are both `LIGO−Raw` which is the type of raw LIGO data to be extracted. $pdata1$, $pdata2$ and $pdata3$ have the same ontological type `LIGO−Pulsar`. Both `LIGO−Raw` and `PulsarData` are the sub-types of `BaseData`.

We now show how the two service types are defined by extending the base transformation type `TR`:

$$\begin{aligned}
&\texttt{TR} \equiv \{\, input :: \texttt{BaseData−Scheme list},\\
&\qquad output :: \texttt{BaseData−Scheme list}\}\\
&\texttt{Extract} \equiv \texttt{TR} + \\
&\qquad \{\, args :: \texttt{String}\,\}\\
&\texttt{Concat} \equiv \texttt{TR} + \\
&\qquad \{\, args :: \texttt{String}\,\}
\end{aligned} \tag{4}$$

where we currently model Grid services with multiple inputs/outputs as having `lists` of input/output data. The reason is that, during the synthesis of workflows, we need to analyse each logical datum of the input/output fields. The datatype of lists has built-in recursively-defined constructors that are convenient for peeling off the elements of a list

The physical properties of Grid services are simply modelled as a string containing the arguments. Currently they are not taken into consideration during the verification and synthesis procedures. We plan to incorporate more logical and physical properties that will be extracted from various application examples, and support pre-conditions that depict the service requirement, local access policy, performance model and economics model, etc.

Below we show $extract1$ as an illustration of service instance declaration:

$$\begin{aligned}
&extract1 :: \texttt{Extract} \equiv \{\\
&\qquad input = [\,\{\, OntType = \texttt{LIGO−Raw},\\
&\qquad\qquad low = 1,\ high = 999\,\}\,],\\
&\qquad output = [\,\{\, OntType = \texttt{LIGO−Pulsar},\\
&\qquad\qquad low = 1,\ high = 999\,\}\,],\\
&\qquad args = \text{“}\cdots\cdots\text{”} \qquad\quad \}
\end{aligned}$$

The $input$ and $output$ fields, inherited from the base type `TR`, semantically describe the abstract data the service instance would require and the data that is produced as a result of the computation performed. In this style, we have the specifications of $extract2$ and $concat$:

$$\begin{aligned}
&extract2 :: \texttt{Extract} \equiv \{\\
&\qquad input = [\,\{\, OntType = \texttt{LIGO−Raw},\\
&\qquad\qquad low = 1000,\ high = 1999\,\}\,],\\
&\qquad output = [\,\{\, OntType = \texttt{LIGO−Pulsar},\\
&\qquad\qquad low = 1000,\ high = 1999\,\}\,],\\
&\qquad args = \text{“}\cdots\cdots\text{”} \qquad\quad \}\\
&concat :: \texttt{Concat} \equiv \{\\
&\qquad input = [\,\{\, OntType = \texttt{LIGO−Pulsar},\\
&\qquad\qquad low = 1,\ high = 999\,\},\\
&\qquad\qquad \{\, OntType = \texttt{LIGO−Pulsar},\\
&\qquad\qquad low = 1000,\ high = 1999\,\}\,],\\
&\qquad output = [\,\{\, OntType = \texttt{LIGO−Pulsar},\\
&\qquad\qquad low = 1,\ high = 1999\,\}\,],\\
&\qquad args = \text{“}\cdots\cdots\text{”} \qquad\quad \}
\end{aligned}$$

4 Grid Workflow Composition

4.1 Formal Specification of Grid Workflow

A workflow, which is a network composed of a number of Grid services, can be modelled as the accumulated data transformation. In this fashion, a Grid service is then a singular workflow that consists of only one high-level service component.

Abstract workflows have only workflow-level properties instead of concrete runtime parameters, such as the execution arguments and physical locations. Therefore we model workflows as generic processes that are of the type of `TR`.

In our implementation, we enrich the TR to have more quality of service (QoS) properties, including provenance, runtime and reliability, which is the probability that the process will execute successfully. QoS information is included for the benefit further work, such as the recently started project "Inferring Quality of Service Properties for Grid Applications", which aims to develop compositional calculi for propagating QoS properties around workflows. TR, the common base type of workflows and processes, is now defined as:

$$\begin{aligned} \texttt{TR} \equiv \{\, & \mathit{input} :: \texttt{BaseData list}, \\ & \mathit{output} :: \texttt{BaseData list}, \\ & \mathit{prov} :: \texttt{String list}, \\ & \mathit{runtime} :: \texttt{Integer}, \\ & \mathit{reliab} :: \texttt{Real} \quad \} \end{aligned} \tag{5}$$

The $prov$ field is designed to record the provenance, which, in the current implementation, is simply constructed by appending together the $prov$ fields of the workflow's components. The runtime field is the sum of the execution times.

4.2 Workflow Verification

So far, we have provided the sequential "$\rightarrow$" and parallel "$\|$" workflow combination operators. So the grammar describing our workflows is defined as:

$$\texttt{WF} \equiv \texttt{TR} \mid \texttt{WF} \rightarrow \texttt{WF} \mid \texttt{WF} \parallel \texttt{WF} \tag{6}$$

The workflow example as shown in Figure 1 can then be formalised using these two operators:

$$\texttt{ligowf} \equiv (\mathit{extract}1 \parallel \mathit{extract}2) \rightarrow \mathit{concat} \tag{7}$$

Recall workflows are composite processes that accumulate data transformations. By observing the overall data transformation, we specify the above workflow as a generic process of type TR. Assuming the provenance fields for $extract1$, $extract2$ and $concat$ are defined as "VO1", "VO1" and "VO2"; the runtimes are 20, 30 and 20; and the reliabilities are 0.80, 0.60 and 0.70, respectively[2], then the resulting combined workflow is:

$$\begin{aligned} \mathbf{spec} \equiv \{\, \mathit{input} &= [\mathit{rdata}1,\ \mathit{rdata}2\,], \\ \mathit{output} &= [\mathit{pdata}3\,], \\ \mathit{prov} &= [\text{“}\mathit{VO}2\text{”},\ \text{“}\mathit{VO}1\text{”},\ \text{“}\mathit{VO}1\text{”}\,], \\ \mathit{runtime} &= 50, \\ \mathit{reliab} &= 0.336 \quad \} \end{aligned} \tag{8}$$

[2]The workflow runtime is obtained by summing the runtime fields on sequentially executed processes and enumerating the maximum runtime of processes executed in parallel.

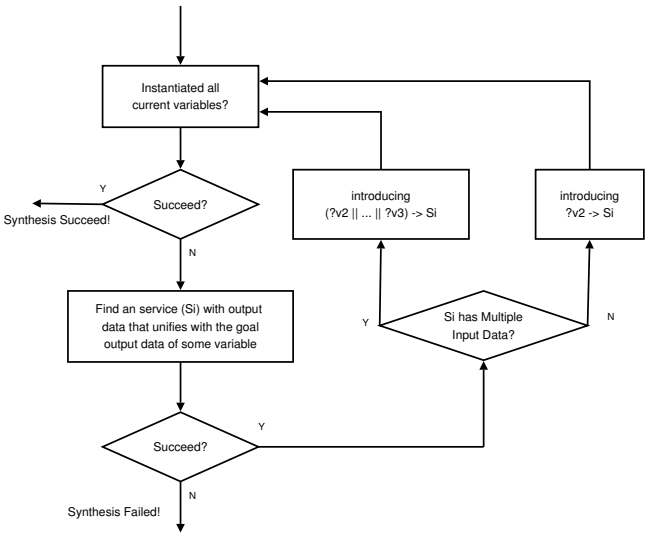

Figure 2. Tactic Control Flow.

We can prove that the workflow logowf satisfies the specification **spec**, by proving:

$$\mathit{satisfies}(\texttt{ligowf}\,, \mathbf{spec}) \tag{9}$$

or in detail, we prove:

$$\begin{aligned} & (\,\texttt{ligowf.input} = [\mathit{rdata}1,\ \mathit{rdata}2\,]\,) \\ & \quad \wedge\, (\,\texttt{ligowf.output} = [\mathit{pdata}3\,]\,) \\ & \quad \wedge\, (\,\texttt{ligowf.prov} = [\text{“}\mathit{VO}2\text{”},\ \text{“}\mathit{VO}1\text{”},\ \text{“}\mathit{VO}1\text{”}\,]\,) \\ & \quad \wedge\, (\,\texttt{ligowf.runtime} \leq 50\,) \\ & \quad \wedge\, (\,\texttt{ligowf.reliab} \geq 0.336\,) \end{aligned}$$

4.3 Workflow Synthesis

Equipped with the formal specifications of the service instances and workflow, we are able to generate the desired workflow from its specification using deductive synthesis.

When a workflow specification is given, such as **spec** defined as in (8), the goal we need to prove is defined in Isabelle notation as:

$$\mathit{satisfies}(?\mathit{wf}\,, \mathbf{spec}) \tag{10}$$

where a variable starting with ? is a schematic variable, which can be instantiated to be any term of the correct type, i.e. $?wf$ is implicitly existentially quantified. The goal can be read as "What kind of workflow can satisfy the specification **spec**?"

The synthesis of this simple workflow is performed by repeatedly applying to the goal the set of tactics given in Figure 2:

The deductive synthesis of $?wf$ from equation (10) can be achieved as follows:

1. Trying to instantiate "$?wf$" directly with criteria:

$$(\,input(?wf) = [\,rdata1,\ rdata2\,]\,) \\ \wedge \quad (\,output(?wf) = [\,pdata3\,]\,)$$

   fails, as there is no service that directly matching these inputs and outputs.

2. Unifying only the output of "$?wf$", $output(?wf) = [\,pdata3\,]$, succeeds with the service $concat$.

3. This introduces the sequential operator, "$\rightarrow$", and the a goal:

$$satisfies(\ ?wf1 \ \rightarrow\ concat, \mathbf{spec}) \qquad (11)$$

   (a) $concat$ has multiple inputs? Yes.

   (b) The parallel operator, "$\|$" is introduced which results in the new goal:

$$satisfies(\ (?wf2 \ \|\ ?wf3) \ \rightarrow\ concat, \mathbf{spec}) \qquad (12)$$

      i. The meta variable "$?wf2$" can now be instantiated with either $extract1$ or $extract2$.

      ii. The meta variable "$?wf3$" can then be instantiated with the other one.

      iii. All schematic variables instantiated, the synthesis succeed.

The tactics are currently applied interactively, but a fully automated version is being developed. Equipped with additional tactics and heuristics, we have successfully synthesised more complex workflows with nested operators, such as the one shown in Figure 3. To synthesise this complex workflow, we just need to build the goal as following:

$$\begin{aligned} satisfies(?wf, \{\, input &= [\,data1,\ data11,\ data20\,], \\ output &= [\,data22\,], \\ prov &= ?prov, \\ runtime &= ?time, \\ reliab &= ?r \quad \}) \end{aligned}$$

The synthesis proof produces the following synthesised workflow:

$$\mathtt{wf} = \Big(\Big(p1 \ \rightarrow\ (p2 \,\|\, p3 \,\|\, p4 \,\|\, p5) \ \rightarrow\ p6\Big) \,\| \\ \Big(q1 \ \rightarrow\ q2 \ \rightarrow\ (q3 \,\|\, q4 \,\|\, q5) \ \rightarrow\ q6\Big) \,\|\, r1\Big) \ \rightarrow\ r2$$

The workflow properties of provenance, runtime and reliability do not contribute to the composition of the workflow, so that their values for the synthesised workflow can be left unknown in the query and can be instantiated during the synthesis by calculation from the corresponding QoS properties of the atomic Grid services from which the workflow is composed.

5 Related Work

Chimera [1] is a virtual data system that can be coupled with distributed "Data Grid" services[7] and the resource-mapping sub-system Pegasus [1] to enable on-demand execution of computation from data queries. The Chimera Virtual Data System models the Data Grid environment using the abstraction of *Virtual Data Language* (VDL) [1], in which Grid services are modelled as data transformations that are specified only by the file names of in/output data.

Pegasus is a planning system to reason about the runtime properties of Grid service instances, map the abstract workflow components to physical resources according to users' and resource providers' preference and generate executable workflows to the Condor workflow interface, DAGMan [17]. Pegasus models Grid services as application-specific planner operators with which it reasons about physical resources to construct a plan of deployment.

The Chimera approach to workflow generation is restrictive in terms of the workflow constructs it is capable of supporting. In contrast, our approach supports a richer language and uses deductive synthesis to incrementally synthesise a workflow that has been proved to meet the specification. Regarding the workflow synthesis as a theorem proving problem offers us a rich set of well-studied constructs, including conditional branching, iteration and recursion, greater expressiveness of preconditions and effects, and many powerful techniques that help us in automating the synthesis. Representing the Grid services and data using an *extensible record* datatype in Isabelle, we intend to address resource deployment as an integrated procedure during the workflow synthesis. Our research group has previously contributed to the deductive synthesis of programs with several successful projects [9, 10]. Because of the more complex representation used, and the extra work involved in proving that the specification is met, we expect the automated workflow synthesis to be slower than planning based approached.

6 Discussion and Future Work

Alternative approaches to automated workflow synthesis use AI plan formation rather than deductive synthesis, [1]. Our future work will focus on those areas where deductive synthesis has potential advantages over plan formation. In particular, planners are restricted in both the language used to describe the preconditions and effects of operators and in

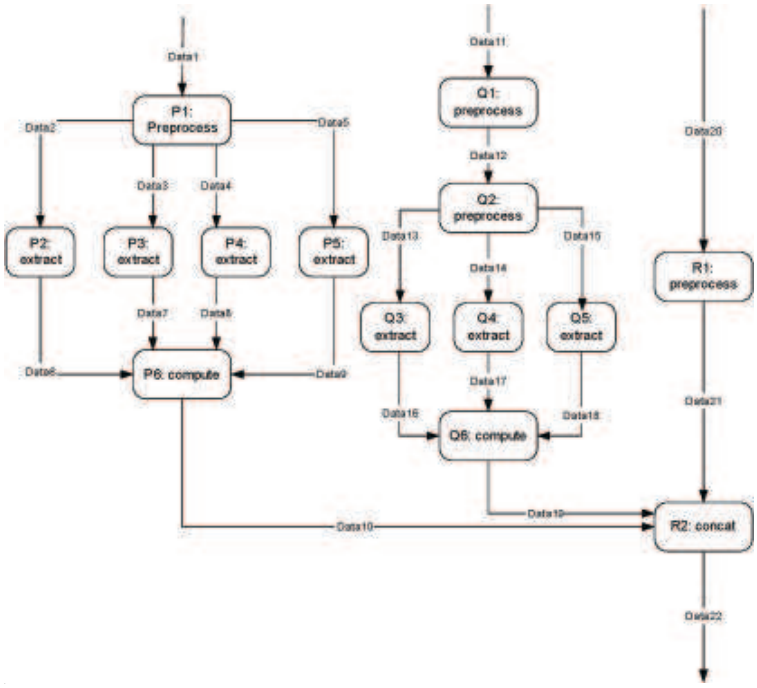

Figure 3. A more complex workflow.

the kinds of workflows they can synthesise. We will investigate whether e-Science requires this greater expressivity and if so demonstrate that deductive synthesis can supply it. We also intend to complete the automation of synthesis.

6.1 Representation of Pre-conditions

Deductive synthesis in Isabelle allows the use of arbitrary, higher-order logical formulae to describe Grid services. In contrast, planners are typically restricted to conjunctions of propositions in describing the pre-conditions and effects of services. We are investigating whether this greater expressivity is of practical use in describing services. For instance, disjunction might be used in pre-conditions to specify that a service can deal with data of several different kinds, but not all kinds; it might be used in effects to explain that a service can have more than one kind of output, perhaps depending on the input. Universal quantification might be used in pre-conditions to ensure that each of a large and varying number of tasks must have certain properties. Existential quantification might be used in service effects to introduce a new object into the ontology, e.g. a new data item. We will both invent services with such properties and search the e-Science literature to find naturally occurring exemplars. We then plan to examine how deductive synthesis can represent and use these services in a natural way, i.e. without the use of kludges to shoe-horn the representation into a conjunction of propositions.

6.2 Synthesis Meeting QoS Requirements

As mentioned earlier, further work also includes trying to infer QoS properties of a compound workflow from its components. For instance, the representation and calculation of QoS properties holds out the prospect of synthesising workflows to meet various QoS properties, e.g. a high reliability or accuracy, or a low runtime. For instance, let us assume there is a "creditability" property of data that indicates how reliable the data replica is. The pre-condition can be specified as follows:

$$\forall\, rdata :: \texttt{BaseData}-\texttt{Scheme}.$$
$$\exists\, pdata :: \texttt{BaseData}-\texttt{Scheme}.$$
$$(credit(rdata) \geq 90\%) \wedge (input(extract1) = rdata)$$
$$\Longrightarrow \; output(extract1) = pdata$$

where $extract1$ requires the input data to have a creditability better than 90%.

6.3 More Workflow Operators

We plan to implement more workflow operators in our framework for better representation of the real Grid environment and more practically useful features. For instance,

the workflow interface of the Unicore system [6] provides supports for conditional branching and iteration constructs. Standard planners support neither conditional branching nor iteration, but deductive synthesis does. Conditional branching is introduced by case splits in the synthesis proof and iteration is introduced by mathematical induction. We intend to investigate whether practical examples of e-Science applications require these constructs. For instance, conditional branching might arise where redundancy is introduced into a workflow by trying first one Grid service and then a mirror service if the first one fails. Thus, the provision of conditional branching in workflows is also required in our synthesis with respect to QoS properties, such as reliability. Iteration might arise where a large and variable number of tasks is to be assigned to a large and variable number of Grid services. The workflow might iterate through lists of tasks and services, making the assignments dynamically. This would enable workflow synthesis to scale-up to applications such as SETI@home. Note that universal quantification might be required to reason about a large and varying number of tasks.

7 Conclusions

In this paper, we present the current progress of our project aiming at the automatic synthesis of e-Science workflows. Our approach, compared with other rival approaches:

- provides richer abstraction of Grid data and services, which will enable on-demand query and computing of complex e-Science data,
- implements a deductive synthesis system in which e-Science workflows are formalised and can be verified against their specification,
- has the potential to provide more expressive descriptions of Grid services and to synthesise more expressive workflows,
- has the potential to represent and reason about QoS properties,
- and is notable as the first attempt, to the authors' knowledge, to apply deductive synthesis technique to synthesise e-Science workflow interactively on users' demand.

We are working to fully automate the synthesis process, to explore the potential for greater expressivity in service descriptions and workflows, and to synthesise workflows with respect to QoS properties. These further work plans dovetail together in that greater expressivity is required for QoS synthesis, and greater expressivity in workflows requires greater expressivity in service descriptions.

References

[1] J. Blythe, E. Deelman, and Y. Gil. Automatically Composed Workflows for Grid Environments. *Intelligent Systems, IEEE*, 2004.

[2] A. Chervenak. Giggle: A Framework for Constructing Scalable Replica Location Services, 2002.

[3] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing, 2001.

[4] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, S. Koranda, A. Lazzarini, and M. A. Papa. From Metadata to Execution on the Grid Pegasus and the Pulsar Search. GriPhyN Technical Report 2003-15, 2003.

[5] E. Deelman, G. Singh, M. P. Atkinson, A. Chervenak, N. P. C. Hong, C. Kesselman, S. Patil, L. Pearlman, and M.-H. Su. Grid-Based Metadata Services. *Proc. SSDBM 2004*, 2004.

[6] D. W. Erwin and D. F. Snelling. UNICORE: A Grid computing environment. *Lecture Notes in Computer Science*, 2150:825, 2001.

[7] A. Ghiselli. DataGrid Prototype 1. *TERENA Networking Conference, 3-6 June*, 2002.

[8] G. Huet. A unification algorithm for typed lambda-calculus. *Journal of Theoretical Computer Science*, 1(1):27–57, 1975.

[9] I. Kraan, D. Basin, and A. Bundy. Middle-out reasoning for logic program synthesis. In *Proc. 10th Intern. Conference on Logic Programing (ICLP '93)* (Budapest, Hungary), pages 441–455, Cambridge, MA, 1993. MIT Press.

[10] D. Lacey, J. Richardson, and A. Smaill. Logic Program Synthesis in a Higher-Order Setting. *Lecture Notes in Computer Science*, 1861:87, 2000.

[11] S. Majithia, D. W.Walker, and W.A.Gray. Automated Composition of Semantic Grid Services. *UK e-Science All Hands Meeting*, 2004.

[12] Z. Manna and R. Waldinger. A Deductive Approach to Program Synthesis. *Journal of Transactions on Programming Languages and Systems*, 2(1):90–121, 1980.

[13] W. Naraschewski and M. Wenzel. Object-Oriented Verification Based on Record Subtyping in Higher-Order Logic. In *Theorem Proving in Higher Order Logics*, pages 349–366, 1998.

[14] L. C. Paulson. Isabelle: A Generic Theorem Prover. *Lecture Notes in Computer Science*, 828, 1994.

[15] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Manohar, S. Patil, and L. Pearlman. A Metadata Catalog Service for Data Intensive Applications. *Proc. ACM/IEEE Supercomputing 2003 Conf. (SC 2003)*, 2003.

[16] R. D. Stevens, A. J. Robinson, and C. A. Goble. myGrid: personalised bioinformatics on the information grid. *Bioinformatics*, 19, 2003.

[17] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. Condor – A Distributed Job Scheduler. In T. Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.

[18] I. Taylor, M. Shields, I. Wang, and R. Philp. Distributed P2P Computing within Triana: A Galaxy Visualization Test Case. To be published in the IPDPS 2003 Conference, April 2003.