Semantic-based On-demand Synthesis of Grid Activities for Automatic Workflow Generation *

Mumtaz Siddiqui, Alex Villazón and Thomas Fahringer
Institute of Computer Science, University of Innsbruck
Technikerstraße 21a, A-6020 Innsbruck, Austria
{mumtaz|avt|tf}@dps.uibk.ac.at

Abstract

On-demand synthesis of Grid activities can play a significant role in automatic workflow composition and in improving quality of the Grid resource provisioning. However, in the Grid, synthesis of activities has been largely ignored due to the limited expressiveness of the representation of activity capabilities and the lack of adapted resource management means to take advantage of such activity synthesis. This paper introduces a new mechanism for automatic synthesis of available activities in the Grid by applying ontology rules. Rule-based synthesis combines multiple primitive activities to form new compound activities. The synthesized activities can be provisioned as new or alternative options for negotiation as well as advance reservation. This is a major advantage compared to other approaches that only focus on resource matching and brokerage. Furthermore, the new synthesized activities provide aggregated capabilities that otherwise may not be possible, leading towards an automatic generation of Grid workflows. We developed a prototype to demonstrate advantages of our approach.

1. Introduction

The Grid enables resource sharing and coordinated problem-solving across computers and humans in a distributed and heterogeneous environment. In such a complex and dynamic environment [1], the challenge lies in coupling resources with activities of potential workflow applications for execution. Enabling scientific workflow applications for the Grid has been identified as an important research topic [2, 3]. An *activity* is an abstraction of a logical resource that refers to as functional description of a software component. A *Grid workflow* is a collection of *activities (tasks)* that may execute across multiple Grid nodes and collectively achieve a common goal. Abstract activities are mapped to concrete deployments at runtime in order to deal with the dynamicity and heterogeneity of Grid [2, 4]. The workflow mapping and execution has been automated [3, 4] whereas the workflow composition is still manual.

Automatic synthesis of Grid activities can play a significant role in automatic composition of workflows and in improving provisioning quality of a Grid resource manager. The *synthesis* involves combining or integrating a set of primitive activities into a single complex activity. Performing such a synthesis is useful in: 1) improving usability of the Grid by aggregating capabilities of underlying resources, 2) generating multiple options built on scarce resources (activities) to be provisioned on-demand by a Grid resource manager, and 3) optimizing resource allocation with adapted planning mechanism. Unfortunately, synthesis of activities in the Grid has been largely ignored due to unobtrusive representation of their capabilities and lack of adapted resource management mechanisms to take advantage of the newly generated activities.

This paper introduces a new mechanism for automatic synthesis of activities by applying ontology rules. An ontology represents an explicit conceptual model with formal logic-based semantics. An extendable foundation ontology is provided in which each primitive activity is modelled as a separate concept in terms of its *inputs*, *outputs*, *usage assumptions* and *after effects* [5]. Ontology rules can be used to defined new concepts and applying them to existing ontology results in new entailment. We introduce new rules for synthesis of activities which integrate primitive activities to form new synthesized activities. These rules are defined by following a set of well defined workflow patterns and applied to the activity knowledge-base with the help of rule-based reasoning tools such as Pellet [6] and Jena [7]. Rules-based semantics gives the declarative descriptions of activities a well-defined meaning by specifying ontological foundations and by showing how such foundations are realized in practice.

We extend our previous work about Grid resource management [4, 8, 9] with semantics. This enables the resource manager to become a smart provisioner that can automatically generate a set of complex activities according to defined rules and deliver them on-demand. These complex activities can be treated as standalone workflows or can be used in new and larger workflows. The workflow can be

*This research is partially supported by the European Union through the IST-034601 edutain@grid and FP6-004265 CoreGRID projects, and by the Higher Education Commission of Pakistan.

generated as a side effect in the form of an abstract workflow. Askalon provides a highlevel Abstract Grid Workflow Language (AGWL) [3] along with a set of tools and services that can be used to visualize workflows, to map them onto the concrete deployments [4] and to execute onto the Grid [3]. Since synthesized activities provide new or aggregated capability of its founding constituents, provisioner becomes a better capacity planner and negotiator. With synthesis, a provisioner gets more to offer possibly with different service qualities than originally without activity synthesis. The provisioner accepts user goals, generates matching workflows, and leaves users to focus on their problems.

We use the Semantic Web technologies such as Ontology Web Language (OWL), SPARQL [1] and Jena APIs for semantically-enabled resource management [9, 3] in the line of the Semantic Grid vision [10, 11, 12].

The rest of the paper is organized as follows: In the next section we discuss the motivation behind our work. Section 3 describes the synthesis model. Section 4 introduces rule-based activity synthesis followed by Section 5 which demonstrates effectiveness of our approach. Section 6 gives overview of related work followed by Section 7 that concludes the paper.

2. Motivation

The Semantic Grid is considered as an extension of the current Grid in which information and services are given well-defined and explicitly represented meaning, enabling better cooperation [11]. Resource representations are enriched with semantics using ontologies and discovered based on highlevel user goals. In the Grid, a goal may be a data set that is to be generated by using some input. The generation may involve a series of activities and aggregated power of computing resources. In the Grid, aggregated capability of activities can be utilized to achieve a goal that otherwise may not be possible especially with needed quality of service (QoS). The activities deployed across multiple computers may provide aggregated capability that leads to utilize aggregated *computing power*. Since activities represent self-contained autonomous software components, the aggregation is possible with synthesis.

Consider that we want to render a movie based on a textual description of its scenes. It is likely that there is no (free) tool available that can perform such task, and even if there is one, the tool may not be able to distribute the calculation of different parts of the movie rendering over different computers (movie rendering is compute-intensive). What we can assume is the existence of some freely available tools that collectively could perform this task. For instance, three tools can be selected: first, *POVray* [13] renders a scene description into a series of PNG frames (still pictures), second, *Png2yuv* that pipes an archive of PNG frames to `stdout` as a *YUV4MPEG2* stream, third, *ffmpeg*

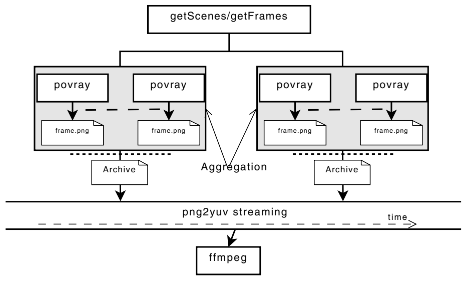


Figure 1. A POVray workflow application.

that convert a stream into MPEG format or display it on a computer screen.

To perform the movie rendering onto the Grid, we can apply the following methods:

- Assuming that all required tools are available on a single Grid node, a user builds a job script and submits to a Grid node by using a middleware operating environment, such as the Globus Toolkit [14].
- If all tools are available in the Grid but not on a single node, then a user composes a workflow and submits to a Grid workflow execution system.

Fig. 1 shows a possible POVray image rendering workflow, where the description of a movie can be separated in several scenes, each scene composed of several frames can be rendered as parallel activities onto the Grid. Finally, all the frames are merged into a *.mpg* movie using a *png2yuv* followed by a *ffmpeg* activity.

Both options described above not only need manual steps for composition, but also require a user to have additional knowledge of each component. For instance, a specific usage of each tool and whether or not several instances of the tool can be executed in parallel. The manual process becomes non-trivial if a user knows its goals but does not know how to achieve efficiently with the Grid. Even a graphical interface may not be helpful without consulting a POVray and a Grid expert. Furthermore, there could be several options [4] to perform the same task with varying QoS. This further jumbles a domain specialist and makes even graphical composition a non-trivial task.

However, with the help of our proposed semantic-based activity synthesis, the entire workflow can be generated automatically. A user specifies its goals, e.g. *a movie in a specific format based on textual description of scenes*. A user may not need to provide input descriptions. Instead, a provisioner can generate multiple options with different types of possible inputs. Since each activity of a workflow can be performed by different tools, this may lead to the generation of multiple alternative options. The provisioner can propose a best option based on a user or system defined criteria.

[1]OWL and SPARQL proposed by W3C (http://www.w3.org)

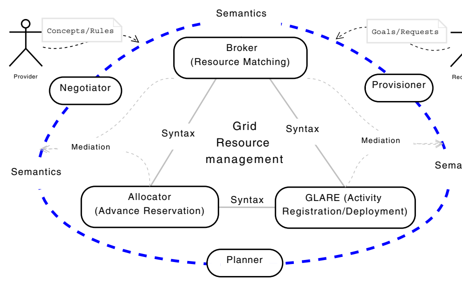


Figure 2. Grid Resource Management Architecture.

Our proposed solution gives explicit meanings to activities and makes them understandable for machine processing. Synthesis rules correlate input and output arguments of activities and combine them to form complex activities that provide aggregated capabilities. For instance, `png2yuv` and `ffmpeg` can be combined to make a compound activity `png2mpeg` whose input becomes input of `png2yuv` and output becomes output of `ffmpeg`.
The synthesis process iteratively generates all possible combinations and ultimately ends up with a workflow that fulfills a user goals. In case of *POVray*, the input of complex resource is a textual description of scenes and output is a MPEG movie.

A provisioner with multiple options becomes better negotiator for resources. It can offer alternative options with varying QoS. For instance, a provisioner may offer following options to fulfill a similar goal:

- A single application or service that is available on a Grid node at a certain timeframe for which a client needs to make advance reservation.
- A set of tools capable of performing same task collectively. The provisioner can generate a workflow and offer it to the client. The client then can contact the workflow enactor service to execute the generated workflow onto the Grid.
- A provisioner can propose to generate a tailor-made service using a wrapper generator [15, 16]. This could be a bit expensive but a client may not care about additional steps of advance reservation or workflow scheduling as the generated wrapper service may work as a dedicated workflow executor.

3. System Model

Askalon's resource management consists of a set of services loosely coupled in a service-oriented fashion. As shown in the Fig. 2, three core services include a *broker* [9] that performs matchmaking of physical resources such as computers, *GLARE* [4] is a Grid application registration, deployment and provisioning framework. This service supports dynamic registration, automatic deployment and on-demand provisioning of primitive activities. The *allocator* is a 3rd service that provides agreement management, negotiation and capacity planning for computing resources [8]. In the proposed extension, these services are wrapped by semantically-enabled services loosely coupled to enable coordinated resource sharing and provisioning with semantics.

3.1. Ontology

We created a basic ontology in OWL-DL using *Protege* [17] It defines fundamental concepts of Grid activities along with physical resources. Advertisement of activities is done by providing semantics descriptions either by using a simple configuration mechanism, or automatically transforming from syntactical descriptions registered in GLARE as an XML-based internal representation of activities. Each activity is modelled as an extension of basic concepts defined in the foundation ontology. These ontologies are asymmetrically extensible so that resource providers can easily extend fundamental concepts without loosing their semantic soundness. They do not have to agree on a certain terminology while extending semantics of generic concepts. For example, if a term `POVray` is a concept of `MovieRenderer` in our foundation ontology, then it can be extended asymmetrically to a new term `MyPOVray`. A reasoner then can automatically and unambiguously infer that `MyPOVray` is a specialization of the `POVray` and `MovieRender`.

Although a reasoner can derive additional ontological entailment, based on property and class hierarchies i.e. subsumptions, sometimes it is necessary to infer pertinent information that cannot be determined otherwise. This emphasizes the need of rules. Since rules are based upon OWL Lite and DL therefore they may have bindings to the underlying ontology. Ontology rules play a vital role in the proposed activity synthesis.

3.2. Activity Synthesis Problem

The synthesis of activities is a problem of combining multiple activities by linking output of one activity to the matching input of an other activity in order to form a new synthesized activity. Let $\mathscr{A}$ be a set of activities involved in the synthesis, $\mathscr{P}$ a set of arguments (like files, integers etc.), then a synthesized activity α is modelled as a triple:
$\alpha = \{\mathscr{I}_\alpha, \mathscr{O}_\alpha, \mathscr{A}_\alpha\}$ where $\alpha \cap \mathscr{A}_\alpha = \oslash \wedge \mathscr{I}_\alpha \cap \mathscr{O}_\alpha = \oslash$ and $\mathscr{I}_\alpha \subseteq \mathscr{P}$ is a set of input arguments of α, $\mathscr{O}_\alpha \subseteq \mathscr{P}$ is a set of output arguments of α and $\mathscr{A}_\alpha \subseteq \mathscr{A}$ is a set of activities which are combined during synthesis to form α. Let $\mathscr{I}_\beta = \bigcup_{\forall \delta \in \mathscr{A}_\alpha} \mathscr{I}_\delta$ is a set of input arguments of all activities $\in \mathscr{A}_\alpha$, $\mathscr{O}_\beta = \bigcup_{\forall \delta \in \mathscr{A}_\alpha} \mathscr{O}_\delta$ is a set of output arguments of all activities $\in \mathscr{A}_\alpha$, and $\mathscr{P}_\beta$ is a set of arguments that are produced and consumed internally by activities $\in \mathscr{A}_\alpha$; if
$\mathscr{P}_\beta = \mathscr{I}_\beta \cap \mathscr{O}_\beta$ then $\mathscr{I}_\alpha = \mathscr{I}_\beta - \mathscr{P}_\beta$ and $\mathscr{O}_\alpha = \mathscr{O}_\beta - \mathscr{P}_\beta$.

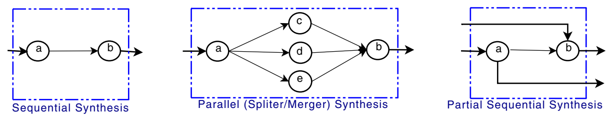

a
b
Sequential Synthesis
a
c
d
e
b
Parallel (Spliter/Merger) Synthesis
a
b
Partial Sequential Synthesis

Figure 3. Sequential and parallel flows patterns.

4. Semantics in the Grid

4.1. Activity Synthesis

The activity synthesis is possible by following workflow patterns. We provide a set of synthesis rules that follow data flow pattern for primitive activities as shown in Fig. 3. These rules are fired iteratively resulting in automatic generation of complex activities. From data flow perspective, both sequential and parallel flows are very important patterns of a workflow.

Sequential Flow

This is one of the basic workflow patterns in which two activities are combined with each other such that output (a set of arguments) of one activity is mapped to the input of second activity. Formally, if we have two activities $\alpha \in \mathscr{A}$ and $\beta \in \mathscr{A}$ such that

$$\alpha = \{\mathscr{I}_\alpha, \mathscr{O}_\alpha, \mathscr{A}_\alpha\} \;\; \beta = \{\mathscr{I}_\beta, \mathscr{O}_\beta, \mathscr{A}_\beta\} \;\; \mathscr{O}_\alpha = \mathscr{I}_\beta \\ \Longrightarrow \gamma = \{\mathscr{I}_\alpha, \mathscr{O}_\beta, \{\alpha, \beta\}\}$$

A new synthesized activity γ pipes output $\mathscr{O}_\alpha$ of activity α to input $\mathscr{I}_\beta$ of activity β. We transform this formal description into ontology concept using following rules:

`[SynthesisRule1` :
(`?a` *input* `?x`)(`?a` *output* `?y1`)
(`?b` *input* `?y2`)(`?b` *output* `?z`)
(`?y1` *rdf:type* `?T`)(`?y2` *rdf:type* `?T`) `makeTemp(?c)`
$\Longrightarrow$ (`?c` *rdf:type* `PipedActivity`)
    (`?c` *input* `?x`)(`?c` *output* `?z`)]

According to this rule a new activity *c* is generated; `makeTemp` is a *built-in* rule that is used to create a new concept in the underlying ontology. `PipedActivity` is a basic concept defined in foundation ontology and `rdf`, `rdfs` and `owl` represent namespace prefixes which are used as default namespace for RDF, RDFS and OWL in the Jena rule-based reasoner. A non-trivial form of sequential flow synthesis is the one in which output $\mathscr{O}_\alpha$ of an activity α matches with input $\mathscr{I}_\beta$ of an other activity β but at least one side always matches partially. In this case, partially matched arguments are consumed internally whereas un-matched arguments become part of input/output of resulting activity γ. Formally, let $\mathtt{D} = \mathscr{O}_\alpha \cap \mathscr{I}_\beta$ then

$$\mathscr{O}_\alpha \cap \mathscr{I}_\beta \neq \oslash \;\wedge\; \mathscr{O}_\alpha \cap \mathscr{I}_\beta \neq \mathscr{O}_\alpha \cup \mathscr{I}_\beta \Longrightarrow \\ \gamma = \{\{\mathscr{I}_\alpha + (\mathscr{I}_\beta - \mathtt{D})\}, \{\mathscr{O}_\beta + (\mathscr{O}_\alpha - \mathtt{D})\}, \{\alpha, \beta\}\}$$

Such kind of synthesis is very useful for generating complex workflows, the following rule makes it possible:

`[SynthesisRule2` :
(`?a` *rdf:type* `GridActivity`)
(`?b` *rdf:type* `GridActivity`)
(`?a` *owl:differentFrom* `?b`)
`partialDataFlow(?a,?b) makeTemp(?c)`
$\Longrightarrow$ (`?c` *rdf:type* `SequentialPairedActivity`)
    `partialDataFlow(?c, ?a, ?b)`]

The `partialDataFlow` is a customized *builtin* implemented in Java and used in Jena library to simplify the reasoning that is otherwise non-trivial to provide in a rule. In the rule body it returns *true* if *a* and *b* has partial output and input match whereas in rule head, it associate input/output arguments to newly created activity *c* accordingly.

Parallel Flow

The true essence of the Grid is to execute at least some of the activities in parallel on different nodes so that a speedup or significant improvement in QoS can be achieved. Achieving a speedup is crucial for scientific application. We define rules that result in the creation of complex activities that can execute in parallel, partially or as a whole, for instance, by following the master/slave pattern. In master/slave pattern an activity α splits a task into sub tasks and distributes them among multiple instances of a slave activity γ. Then a third activity β collects partial results from slaves and consolidate in final result. Since we assume that all activities are autonomous and self-contained therefore the *splitter*, *merger* and *slave* activities are disjoint activities, that leads to:

$$\alpha = \{\mathscr{I}_\alpha, \mathscr{O}_\alpha, \mathscr{A}_\alpha\} \;\; \beta = \{\mathscr{I}_\beta, \mathscr{O}_\beta, \mathscr{A}_\beta\} \;\; \gamma = \{\mathscr{I}_\gamma, \mathscr{O}_\gamma, \mathscr{A}_\gamma\} \\ \mathscr{O}_\alpha = \mathscr{I}_\gamma \wedge \mathscr{I}_\beta = \mathscr{O}_\gamma \Longrightarrow \delta = \{\mathscr{I}_\alpha, \mathscr{O}_\beta, \{\alpha, \beta, \gamma\}\}$$

which can be translated to ontology rules as:

`[SynthesisRule3` :
(`?a` *output* `?x`)(`?x` *rdfs:subClassOf* `all(argument ?y1)`)
(`?b` *input* `?z`)(`?z` *rdfs:subClassOf* `all(argument ?y2)`)
(`?a` *input* `?ain`)(`?b` *output* `?bout`) (`?c` *input* `?y1`)
(`?c` *output* `?y2`)`makeTemp(?d)`
$\Longrightarrow$ (`?d` *rdf:type* `MasterSlaveActivity`)
    (`?d` *splitter* `?a`) (`?d` *merger* `?b`)(`?d` *slave* `?c`)
    (`?d` *input* `?ain`) (`?d` *output* `?bout`)]

In an other form of parallel flow, any two activities α and β can be connected though a set of slave activities $\mathscr{A}_s$ so that $\mathscr{O}_\alpha$ is connected to the $\mathscr{I}_\beta$ by activities $\in \mathscr{A}_s$.

$$\alpha = \{\mathscr{I}_\alpha, \mathscr{O}_\alpha, \mathscr{A}_\alpha\} \;\; \beta = \{\mathscr{I}_\beta, \mathscr{O}_\beta, \mathscr{A}_\beta\} \;\; \mathscr{A}_s \in \mathscr{A} \\ \mathscr{O}_\alpha = \bigcup_{s \in \mathscr{A}_s} \mathscr{I}_s \;\wedge\; \mathscr{I}_\beta = \bigcup_{s \in \mathscr{A}_s} \mathscr{O}_s \\ \Longrightarrow \gamma = \{\mathscr{I}_\alpha, \mathscr{O}_\beta, \{\{\alpha, \beta\} \cup \mathscr{A}_s\}\}$$

This is a well known workflow pattern. As shown in Fig. 3 *(parallel flow)*, activities *a* and *b* work as splitter and merger

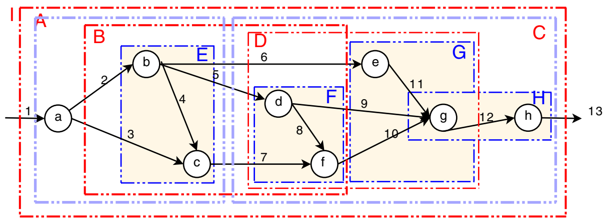


Figure 4. A Sample Synthesis of activities.

respectively. Each activity $c_i \in \{c_1, ..., c_n\}$ works as a slave and can be executed in parallel.

If $\mathscr{O}_\alpha = \{o\}$ i.e. a collection of similar arguments and $\mathscr{I}_\beta = \{i\}$ an other collection of similar arguments then slave activities $\mathscr{A}_s = \{s\}$ i.e. they are instances of same activity and thus synthesized activity can be composed in a parallel loop, for instance `parallelFor` that is a highlevel construct in AGWL. If an activity provider adds some information about activity usage, such as *Parallelizable* and *Iteratable*, then synthesis may lead to repeatable control flow activities.

Fig. 4 depicts a set of primitive activities $\mathscr{A} = \{a - h\}$, a set of arguments $\mathscr{P} = \{1 - 13\}$. All primitive activities are independently defined. The flow of arguments is formed after synthesis. First primitive activities are combined in pairs (shown as shaded rectangles) and then more complex activities are formed (shown as rectangles). Synthesized activities are labeled as $\{A - H\}$. The most complex activity is I that transforms argument 1 into argument 13. After the activity synthesis, compound activities are formed as follows:

$$\begin{aligned}
A &= \{\{1\},\{5,6,7\},\{a,E\}\} &&= \{\{1\},\{5,6,7\},\{a,b,c\}\}\\
B &= \{\{2,3\},\{9,10\},\{E,F\}\} &&= \{\{2,3\},\{9,10\},\{b,c,d,f\}\}\\
C &= \{\{5,6,7\},\{13\},\{D,h\}\} &&= \{\{5,6,7\},\{13\},\{d,e,f,g,h\}\}\\
D &= \{\{5,6,7\},\{12\},\{F,G\}\} &&= \{\{5,6,7\},\{12\},\{d,e,f,g\}\}\\
E &= \{\{2,3\},\{5,6,7\},\{b,c\}\} &&= \{\{2,3\},\{5,6,7\},\{b,c\}\}\\
F &= \{\{5,7\},\{9,10\},\{d,f\}\} &&= \{\{5,7\},\{9,10\},\{d,f\}\}\\
G &= \{\{6,9,10\},\{12\},\{e,g\}\} &&= \{\{1\},\{13\},\{e,g\}\}\\
H &= \{\{9,10,11\},\{13\},\{g,h\}\} &&= \{\{9,10,11\},\{13\},\{g,h\}\}\\
I &= \{\{1\},\{13\},\{A,C\}\} &&= \{\{1\},\{13\},\{a,b,c,d,e,f,g,h\}\}
\end{aligned}$$

This shows that synthesis of activities significantly increases total number of activities. The increase in activity search space may result in query hits that otherwise may not be possible.

4.2. On-demand Provisioning

The provisioning is a process of delivering matching resources (activities), including complex activities, on-demand. The resources are matched with user goals along with possible inputs and other QoS parameters. We propose to use SPARQL query language for defining goals as it is a proposed standard query language for ontologies and well supported by various tools, APIs and can be used to define constraints along with rules. A goal in the form of SPARQL query looks like:

```
SELECT ?a
WHERE {?a rdf:type MovieRenderer .
  ?a outout ?out .
  ?out rdf:type MPEGFile .
}
```

In order to address multiple constraints while generating complex activities, we introduce constraints in the ontology rules. For intstance, a complex activity should be generated only if average reliability of primitive activities is at least 0.5%. This is however, not trivial with current rule language [18]. To overcome this limitation, we propose to use *built-ins* in rules.

Built-ins and Constraints

Built-ins constructs in a rule language is a modular approach of adding new entailment that otherwise is not possible with current reasoners. A set of *built-ins* are available for Jena that can be executed on underlying ontology and knowledge-base. The set includes built-ins for mathematical calculations, comparisons, boolean evaluations, string and collection manipulations, etc. A set of custom *built-ins* is introduced that augments process of rule-based entailment. Customized *built-ins* can also be provided such as the following rule that generate an activity if and only if average reliability of primitive activities is more than 0.5.

```
[SynthesisRuleReliableActivity :
  (?a input ?x)(?a output ?y)(?b input ?y)
 (?b output ?z)(?a reliability ?i) (?b reliability ?j)
 average(?i,?j,?av) greaterThan(?av,0.5)
 makeTemp(?c)
==> (?c rdf:type ReliableActivity)
    (?c input ?x)(?c output ?j)
    ?c reliability ?avg) ]
```

Similarly, user goals may also contain constraints in the form of filters as part of SPARQL query.

```
SELECT ?a
WHERE {?a rdf:type MovieRender .
  ?a outout ?out .
  ?out rdf:type MPEGFile .
  ?a reliability ?reliability .
  ?a animation ?animation .
  ?animation rdf:type CyclicAnimation .
  FILTER (?reliability >= 0.5)
}
```

A custom built-in called *agwl* is provided, once user goals are mapped to at least one possible candidate activity then *agwl* is fired that transforms a matching synthesized activity into an AGWL document as a side effect.

The built-in *rpdp* is a reservation policy decision point that optionally works as a part of *assumptions* and interact with Askalon reservation service [8] for user authorization to check if it has advance reservation.

Assumptions and Effects

As described in Section 3.1, an activity or application provider may have some assumption about the user or a user may assume something about providers. Similarly, there could be some *after-effects* once an activity is executed. An assumption and effects described in primitive activities are aggregated in compound activities. These assumption needs to be true before activity execution and effects may be exercised after activity execution. For instance, an activity provider may assume that a user have advance reservation for activity execution on a certain Grid node and user account is charged as an after-effect.

[`SynthesisRuleResAssumption` :
... ...
(`?a` *rdf:type* `AGridActivity`)
(`?a` *hasAssumption* `?r`) (`?r` *rdf:type* `Reservation`)
(`?r` *owner* `?o`) `isCaller(?o)` $\Longrightarrow$ ...]

[`SynthesisRuleChargingEffect` :
... ...
(`?u` *rdf:type* `sg:AGridUser`)
(`?u` *hasExecuted* `?a`) (`?a` *hasAssumption* `?r`)
(`?r` *rdf:type* `Reservation`) (`?r` *owner* `?o`)
$\Longrightarrow$ `charge(?o,?r)`]

Rules can be included in the ontology as a file or URL. This enables application providers to update them dynamically as well as remotely.

5. Discussion and Experiments

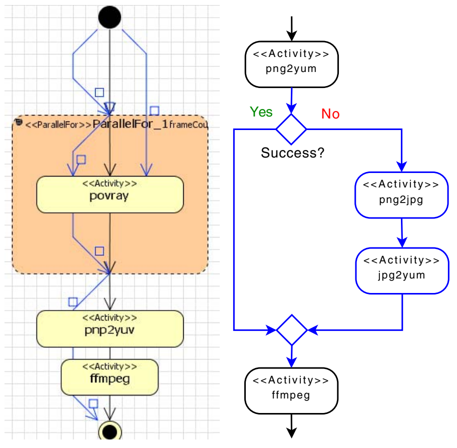
We designed our foundation ontology using Protege-OWL editor, and implemented prototype of the proposed activity synthesis mechanism using Jena APIs. Jena provides Java APIs for dynamically creating new ontological concepts, populating knowledge base and rule-based reasoning. Based on the foundation ontology, an activity (application) provider can add new concepts and provides semantic description of their resources. This can be done either using Protege ontology editor or our simplified configuration description mechanism as defined in our previous work about activity registration and deployment mechanism [4, 3].

Workflow Generation: In order to demonstrate an automatic workflow generation with synthesis of activities, we independently register 3 tools as primitive activities as described below in our triple notation i.e. $\alpha = \{\mathscr{I}_\alpha, \mathscr{O}_\alpha, \mathscr{A}_\alpha\}$:

```
povray = { {InitializationFile,ArgumentFile,
  SceneDescriptionFile,StartFrameInt,
  FramesCountInt,TotalFramesInt}
  {PNGArchiveFile}{∅} }
png2yuv = { {PNGArchiveFile}{YUVStream}{∅} }
ffmpeg = { {YUVStream}{MPEGFile}{∅} }
```

An ontology generator component reads the descriptions, generates ontologies and registers in the knowledgebase. All these concepts including activities and argument are defined as distinguished OWL concepts. After applying the synthesis rules and making a query with goal as:

SELECT ?a
WHERE {`?a` *input* `?x` .
  `?a` *outout* `?y` .
  `?x` *rdf:type* `SceneDescriptionFile` .
  `?y` *rdf:type* `MPEGFile` . }



(a) Generated AGWL (b) Control Flow

Figure 5. POVray Workflow

we managed to generate a set of *agwl* documents that can be visualized in Askalon graphical workflow composition tool. One of the generated workflow is shown in Fig. 5(a).

Provisioning Improvement: An interesting aspect of synthesis is that it produces more options for provisioner for selection or negotiation. For instance, consider three activities α, β and γ as described below:

$$\alpha = \{\mathscr{I}_x, \mathscr{O}_y, \mathscr{A}_\alpha\} \ \beta = \{\mathscr{I}_y, \mathscr{O}_z, \mathscr{A}_\beta\} \ \gamma = \{\mathscr{I}_x, \mathscr{O}_z, \mathscr{A}_\gamma\}$$

By applying synthesis rules, beside others a complex activity δ is generated where $\delta = \{\mathscr{I}_x, \mathscr{O}_z, \{\alpha, \beta\}\}$ is a combination of first two activities α and β as $\mathscr{O}_\alpha = \mathscr{I}_\beta$. As the synthesized activity δ is same as γ the provisioner gets two options to offer instead of just one as it was the case before synthesis. Furthermore, in case if there is no deployment of γ in the Grid, the provisioner may select/offer δ as an alternative option or vice versa, i.e. γ as a replacement of δ.

By following this approach, activity *png2yum* in Fig. 5(a) can be replaced with following two activities:

```
png2jpg={{PNGArchiveFile}{JPGArchiveFile}{∅}}
jpg2yum={{JPGArchiveFile}{YUVStream}{∅} }
```

Similar to *png2yuv* both *png2jpg* and *jpg2yum* transform *PNGArchiveFile* to the *YUVStream*. This may lead to the generation of a complex workflow with an alternative control-flow as shown in Fig. 5(b), that means if *png2yuv* does not succeed then control moved towards *png2jpg* followed by *jpg2yum*.

Throughput: In Fig. 6 we compare average throughput of different types of queries for available activities. A query for a primitive activity, a query for a complex activity generated with simple combination of primitive activities with single input and output, and a query for a complex activity generated after complex synthesis of activities with partially matching arguments). It is depicted that overhead of queries for synthesized activities is very small if time is measured for activity matching excluding synthesis overhead.

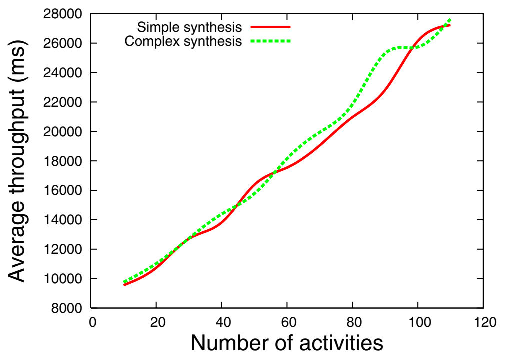


Figure 6. Average response time per transaction.

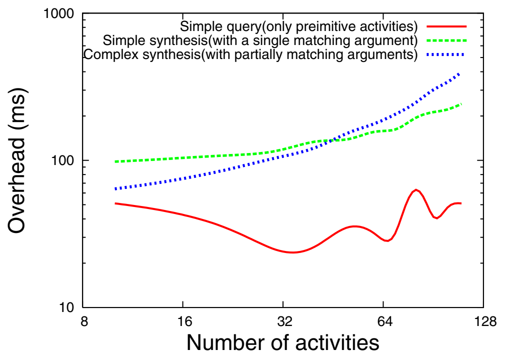


Figure 7. Synthesis overhead.

Overhead: Fig. 7 compares average overhead of synthesis rules when applied dynamically with the query. The overhead is about $10-30s$ for varying number of activities. This is due to the fact that rules are fired iteratively for new entailment and process stops only when further entailment is not possible. Overhead of complex combinations is slightly higher than simple combinations. However, for a few hundred activities it is just under $30s$ that is negligible compared with manual composition time.

Firing rules: The introduction of a new rule in an ontology starts an iterative entailment process that stops when new entailment become impossible. This is called forward chaining and is considered an inefficient approach. However, iteration is necessary for determining interdependency of multiple rules and thus generating all possible combination of primitive and new activities. This results in faster subsequent queries as facts in the knowledge base are not required to be recalculated. Thus, user goals are efficiently mapped to an already generated workflow. A backward chaining is also possible in which only required entailment are generated. This is an efficient entailment process but the rules are fired for each new instance. This slowed down query response time.

6. Related Work

Numerous researchers are working in the area of semantic Grid [11, 12, 19]. However, most of the work has been focused on the problem of matchmaking of physical resources [20]. Similarly, the work in [21] provides a mechanism to map a small subset of Unicore resource descriptions with GLUE schema using ontologies.

Web service composition is similar to Grid workflow composition and activity synthesis. In the area of Semantic Web, a lot of conceptual work has been done but it does not focus on legacy applications. One such example is WSMO [22] that provides an ontology-based description mechanism only for Web services. The work described in [23] applies planning to the problem of web service composition.

In the area of Semantic Grid very few researchers has been working on automatic composition of workflows. The work in [24] addresses problem of machine-assisted composition of workflows for E-Science data using deductive synthesis. The role of planning is discussed in [25] for Pegasus. The authors propose a workflow planner that uses heuristic control rules and searches a number of alternative pre-built complete plans in order to find better quality plan.

A semi-automated approach is introduced in [26] for knowledge evolution of ontology schemas that is applied to the process of new Grid service registration. In K-WfGrid project, knowledge evolution supporting automatic workflow composition has been studied using ontology alignment methods and Petri-net based Grid workflows [26].

In contrast, we introduce a semantic-based on-demand synthesis of Grid activities to form complex activities that leads to on-demand generation of abstract workflow applications that can be concretize automatically at runtime. The synthesized activities provide an entirely new capability, an alternative option of an existing activity and/or an aggregated capability of its basic building blocks. Furthermore, our approach exploits semantic-based synthesis of activities that improves provisioning and brokerage capability of resource management for the Grid.

7. Conclusion

In this paper we formalized the problem of the Grid activity synthesis. Grid activities are software components (executables/services) which are described in terms of abstract and concrete descriptions. We propose a rule-based synthesis mechanism that can be used to combine multiple activities to form compound activities. A set of custom *built-ins* is introduced that augments process of rule-based entailment. The synthesized activities either provide an entirely new or an aggregated functionality of combined activities, or provide alternative options with different quality of service. This synthesis of activities leads to automatic generation of Grid workflows that can be offered on-demand and mapped to the Grid dynamically at runtime. Furthermore, the synthesis generates more (compound) activities thus increases activity search space and result in improved negotiability of a provisioner. That means a provisioner gets more options to offer than it originally may have.

We demonstrated the effectiveness of the synthesis with examples and experiments. We plan to introduce more rules and generate more complex workflows with different types of control and data flows. We also plan to analyze possible improvement in query-hits with synthesis.

References

[1] D. De Roure, N.R. Jennings, and N.R. Shadbolt, "The semantic grid: A future e-science infrastructure," in *Grid Computing - Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and A. J. G. Hey, Eds., pp. 437–470. John Wiley and Sons Ltd., 2003.

[2] Ewa Deelman et. al., "Mapping abstract complex workflows onto grid environments," *Journal of Grid Computing, LNCSD9, ISSN 1570-7873*, vol. 1, pp. 25–39, 2003.

[3] Thomas Fahringer, Radu Prodan, Rubing Duan, Juergen Hofer, Farruch Nadeem, Fracesco Nerieri, Stefen Podlipnig, Jun Qin, Mumtaz Siddiqui, H.-L. Truong, Alex Villazon, and Marek Wieczorek, "Askalon: A development and grid computing environment for scientific workflows," *Workflows for eScience, Scientific Workflows for Grids*, p. 450.

[4] Mumtaz Siddiqui, Alex Villazon, Juergen Hofer, and Thomas Fahringer, "GLARE: A grid activity registration, deployment and provisioning framework," in *International Conference for High Performance Computing, Networking and Storage (SuperComputing), SC05*, ACM, Ed., Seattle, Washington, USA, November 12-18 2005, p. 52, ACM Press, ISBN 1-59593-061-2/05/0011.

[5] W3C, "Owl-s: Semantic markup for web services," http://www.w3.org/Submission/OWL-S.

[6] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz, "Pellet: A practical owl-dl reasoner," *Journal of Web Semantics*, p. Available as http://www.mindswap.org/papers/PelletJWS.pdf, 2006.

[7] Hewlett-Packard Development Company, "Jena: A semantic web framework for java," http://jena.sourceforge.net.

[8] Mumtaz Siddiqui, Alex Villazon, and Thomas Fahringer, "Grid capacity planning with negotiation-based advance reservation for optimized qos," in *International Conference for High Performance Computing, Networking and Storage (SuperComputing), SC06*, ACM, Ed., Tampa, Florida, USA, November 11-17 2006, p. 103, ACM/IEEE.

[9] Mumtaz Siddiqui and Thomas Fahringer, "Semantically-enhanced on-demand resource provision and management for the grid.," *Journal of Multiagent and Grid Systems*, vol. 3, 2007.

[10] De Roure D., N.R. Jennings, and N.R. Shadbolt, "The semantic grid: Past,present, and future," in *Proceedings of the IEEE*, March 2005, vol. 93, pp. 669–681.

[11] D. De Roure, J. Frey, D. Michaelides, and K. Page, "The collaborative semantic grid," in *In Proceedings of 2006 International Symposium on Collaborative Technologies and Systems*, Las Vegas, USA, 2006, pp. 411–418, Smari, W. W. and McQuay, W.

[12] Oscar Corcho, Pinar Alper, Ioannis Kotsiopoulos, Paolo Missier, Sean Bechhofer, and Carole Goble, "An overview of s-ogsa: A reference semantic grid architecture," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 4, pp. 102–115, June 2006.

[13] "Persistence of vision raytracer," http://www.povray.org.

[14] The Globus Alliance, "Globus toolkit," http://www.globus.org/toolkit.

[15] Tristan Glatard, David Emsellem, and Johan Montagnat, "Generic web service wrapper for efficient embedding of legacy codes in service-based workflows," in *Grid-Enabling Legacy Applications and Supporting End Users Workshop (GELA'06)*, Paris, France, June 2006, pp. 44–53.

[16] T. Delaittre, T. Kiss, A. Goyeneche, G. Terstyanszky, Winter S., and P. Kacsuk, "Gemlca: Running legacy code applications as grid services,," *Journal of Grid Computing*, vol. 3, no. 1-2, pp. 75–90, 2005.

[17] Stanford Univesity, "Protege: a free, open source ontology editor and knowledge-base framework," http://protege.stanford.edu.

[18] W3C, "Swrl: A semantic web rule language combining owl and ruleml," http://www.w3.org/Submission/SWRL.

[19] Carole Goble, Oscar Corcho, and Wei Xing, "Acton: A semantic information service for egee," in *The 8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, Austin, Texas, 2007.

[20] H. Tangmunarunkit, S. Decker, and C. Kesselman., "Ontology-based Resource Matching in the Grid–The Grid meets the Semantic Web.," in *Second International Semantic Web Conference, Sanibel-Captiva Islands, Florida*, pp. 706–721. Oct. 2003.

[21] John Brooke, Donal Fellows, Kevin Garwood, and Carole Goble, "Semantic matching of grid resource descriptions," *Grid Computing, Lecture Notes in Computer Science*, vol. Volume 3165/2004, pp. 240–249, 2004.

[22] Jos de Bruijn, Holger Lausen, Reto Krummenacher, Axel Polleres, Livia Predoiu, Michael Kifer, and Dieter Fensel, "The WSML Family of Representation Languages," Working draft, Digital Enterprise Research Insitute (DERI), March 2005, Available from `http://www.wsmo.org/TR/d16/d16.1/v0.2/`.

[23] D McDermott, "Estimated-regression planning for interactions with web services," in *Sixth International Conference in AI Planning Systems,*. 2002, Telescience.

[24] B. Yang, A. Bundy, A. Smaill, and L. Dixon, "Deductive synthesis of workflows for e-science," in *SIGAW Workshop of CCGrid 2005, Cardiff*, 2005.

[25] Gil Y., Deelman E., Blythe J., Kesselman C., and Tangmunarunkit H., "Artificial intelligence and grids: workflow planning and beyond," *Intelligent Systems, IEEE*, vol. 19, no. 1, pp. 26– 33, 2004.

[26] Renata Slota, Joanna Zieba, Bartosz Kryza, and Jacek Kitowski, "Knowledge evolution supporting automatic workflow composition," in *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, Washington, DC, USA, 2006, p. 37, IEEE Computer Society.