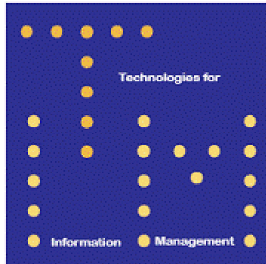

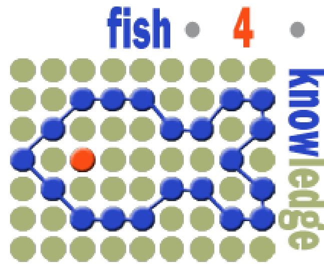

Fish4Knowledge Deliverable D3.2

Process Planning and Composition

Principal Author: UEDIN Workflow
Contributors: G. Nadarajan, Y.-H. Chen-Burger
Dissemination: PU

Abstract:
The workflow component of the F4K project is responsible for the composition and execution of a set of video and image processing (VIP) modules on high performance computing (HPC) machines based on user requirements and descriptions of the video data. It interprets the user requirements as high level VIP tasks, creates workflows based on the procedural constraints of the VIP modules, invokes and manages their execution in the HPC (distributed) environment.
This report emphasises the workflow composition aspect which hinges on AI planning, assisted by the ontologies that were developed for the project. Hierarchical planning models the VIP tasks using a decomposition based approach, and utilises preconditions, effects and postconditions as means to select appropriate steps to solve VIP tasks. Sequences of steps are composed and instantiated as VIP workflows that will be sent for execution.
A baseline workflow manager that oversees the management of queries, such as breaking them into low level video tasks, creating appropriate database entries, communicating with the resource scheduler and front end program is also described. The baseline workflow prototype is a crucial step towards achieving integration in the back end of the F4K system. More effort will be dedicated to the development of the workflow manager to deal with the complex management of tasks and communication with other components.

Deliverable due: Month 18

 © Fish4Knowledge Consortium, 2012

1 Introduction

The pervasiveness of video data has proliferated the need for more automated and scientific methods to store and analyse them. In particular, continuous collection of video data *e.g.* surveillance videos and satellite images, are of major concern because they cause the accumulation of 'big data'. In Fish4Knowledge (F4K), hundreds of underwater video clips are collected daily by NCHC [11] and made available to marine biologists for long-term monitoring, so that marine biologists can make observations such as marine life population changes and behaviours. However, pure manual based observation and analysis is unfeasible; one minute's video clip requires approximately 15 minutes' human effort on average for basic processing tasks which include viewing the clip, labelling interesting frames, adding brief annotation and basic classification of the clips [6]. Currently, continuous video recording has been collected over the past 5 years, and it is rapidly increasing, reaching 100 Terabytes in the near future. We attempt to efficiently store, analyse and communicate these videos and corresponding analysed results to marine biologists using computational-assisted solutions in the form of user interfaces, image processing tools, databases, high performance computing and workflow technologies. In particular, we will be deploying a flexible workflow framework that will enable us to talk with a sophisticated front end user system that manages complex user queries. Based on user queries that are translated to workflow queries, this workflow system will retrieve appropriate video and image analysis modules and run them in high performance computing facilities. In this report, we report our progress so far on the design and development of this workflow framework and prototype.

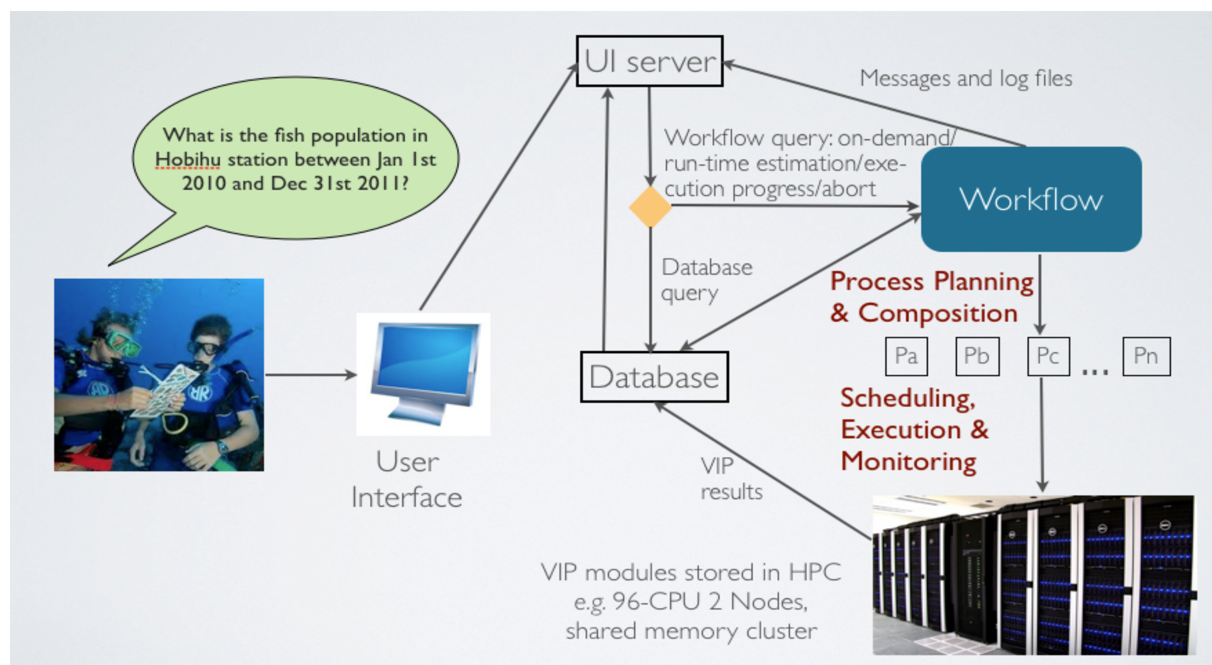

Figure 1: The F4K's workflow component binds high level workflow queries from the user interface to low level image processing components via process planning and composition. It also schedules and monitors the execution of the video processing tasks on a high performance computing environment and reports feedback to the user interface component.

 © Fish4Knowledge Consortium, 2012

The workflow component of the F4K project is responsible for the composition and execution of a set of video and image processing (VIP) modules on high performance computing (HPC) machines based on user requirements and descriptions of the video data. It interprets the user requirements (from the User Interface component) as high level VIP tasks, creates workflows based on the procedural constraints of the modules (Image Processing components) to ultimately invoke and manage their execution in a distributed environment (HPC component). The HPC environment is provided and run remotely at the National Center for Higher Performance Computing (NCHC), Taiwan. Fig. 1 summarises the high level aims of the workflow component, in line with Deliverable 5.1(Component Interface and Integration Plan).

First, a set of requirements for F4K's workflow component is outlined (Section 2). Then, an introduction and a brief overview of existing workflow composition mechanisms will be provided in Section 3, followed by the compute environment available to us (Section 4). The workflow design framework is described in Section 5. It highlights the process planning and composition which relies on ontologies that were developed for the project (available in Deliverable 3.1). The implementation of the baseline workflow prototype is explained in Section 6 which leads to issues to be addressed and progression to the next stage (Section 7).

2 Requirements

A broad set of requirements for F4K's workflow component is formulated to address the problem of automatic video analysis for image processing-naive users (*i.e.* marine biologists) within F4K. These requirements are outlined below:

1. Process Automation
   Clearly, some form of automated assistance would reduce the processing time taken by traditional manual processing considerably. In particular, for large sets of data, manual processing alone would be unfeasible.

2. Rich Process Modelling (Iterative Processing, Conditional Branching, *etc.*)
   Videos are made up of sequences of images or frames. Often, analysis would involve some sort of computation over all the frames of a video. At times, different types of computations are performed on a frame due to some conditions, such as user preferences or existing domain descriptions, such as lighting conditions and murkiness of water. So, there should be mechanisms to cater for different requirements and to address dynamic varying conditions.

3. Performance-Based VIP Tool Selection
   In general, many VIP algorithms exist to perform the same family of tasks. For instance, there are many ways in which a background model (*e.g.* Gaussian mixture model, Intrinsic model) can be constructed. There should be a mechanism that can select the optimal background model algorithm for a given video.

4. Speed *vs.* Accuracy
   An ideal system would be one that is generic in architecture that balances speed of the processing with accuracy of the results. The accuracy has to do with selecting VIP modules that produce the results closest to ground truth values, and speed is related to minimising latency in order to meet on-demand queries by relying on HPC resources.

 © Fish4Knowledge Consortium, 2012

From the requirements we stipulate that an integrated approach combining several traits would be the most suitable for our workflow component. We considered AI planning and ontologies for this purpose because ontologies are generally useful for representation and inference in many applications today while planning technologies have been successful at solving complex problems with well-defined goals using action sequences.

3 Background

This sections provides a brief introduction to workflow and an overview of workflow composition mechanisms in existing systems. For a more thorough treatment of workflows, the reader is referred to the workflow patterns web page [27]. References and overview of major workflow systems such as Pegasus, Triana, Kepler and Taverna can be found in [9].

3.1 An Introduction to Workflow

A workflow is defined as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules"[1]. In other words a workflow consists of all the steps and the orchestration of a set of activities that should be executed in order to deliver an output or achieve a larger and sophisticated goal. A workflow can be seen as a set of activities stored as a model that describes a real world process. Work passes through the model from start to finish, and activities might be executed by people or by system functions. A workflow provides a way of describing the order of execution and dependent relationships between pieces of short-running or long-running work. A workflow enactor or engine manages and executes models of processes. These models can be created and edited by users who are inexperienced in programming. The flow of information, tasks and interpretation of events are facilitated by the workflow enactor. Among the tangible benefits offered by workflows to an organisation include reduced operating costs, improved productivity and faster processing times [1].

A workflow normally comprises a number of logical steps or functional units, referred to as tasks. A task corresponds to a single unit of work. Tasks are connected in the form of a directed graph and can be primitive or non primitive. Primitive tasks are steps that perform single units of work while non primitive ones manage a set of child tasks. Tasks can also represent logical control structures that define scope and direct the execution flow of the workflow, much as code logic controls, such as `if-then` and `while` loops, control the program flow in code.

Workflows may be represented in many forms. As mentioned earlier, workflows are essentially a series of functional units and the dependencies between them define the order in which the units must be executed. Among the well-known models that have been used as the basis for workflow representation languages are Petri nets [16], directed graphs [3], Unified Modelling Language (UML) [5] and Business Process Modelling Notation (BPMN) [29].

With the advent in distributed platforms such as the Grid or e-Science [12] over the past two decades, several workflow management systems have been deployed in distributed platforms. The goal of e-Science workflow systems is to provide a specialised programming environment to simplify the programming effort required by scientists to orchestrate a computational science experiment. Therefore, Grid-enabled systems must facilitate the **composition** of multiple

[1]Workflow Management Coalition: http://www.wfmc.org

 © Fish4Knowledge Consortium, 2012

(software) resources, and provide mechanisms for creating and enacting these resources in a distributed manner.

3.2 Workflow Composition Mechanisms

Studies on workflow systems have revealed four aspects of the workflow lifecycle – composition, mapping (onto resources), execution and provenance capture [9]. We focus on the composition and execution aspects of the workflow lifecycle. Workflow composition can be textual, graphical or semantics-based. Textual workflow editing requires the user to describe the workflow in a particular workflow language such as BPEL [2], SCUFL [25], Condor DAGMan [26] and DAG with XML description (DAX) [10]. This method can be extremely difficult or error-prone even for users who are technically adept with the workflow language.

Graphical renderings of workflows such as those utilised by systems such as Triana, Kepler and VisTrails are easy for small sized workflows with fewer than a few dozen tasks. However many e-Science and video processing workflows are more complex. Some workflows have both textual and graphical composition abilities. The CoG Kit's Karajan uses either a scripting language, GridAnt or a simple graphical editor to create workflows.

Some effort in automatic workflow generation has been undertaken in order to ease the tediousness of manual composition. Planning technology is used to analyse, verify and correct partial workflows in order to perform interactive workflow composition in Composition Analysis Tool (CAT) [18] used by Pegasus. Wings [15] extends this by dealing with the creation and validation of very large scientific workflows. However, CAT requires the user to construct a workflow *before* interactively verifying it to produce a final workflow. Our work, in contrast, aims to construct the workflow interactively or automatically.

Blythe *et al.* [4] have researched into a planning-based approach to workflow construction and of declarative representations of data shared between several components in the Grid. This approach is extendable to be used in a web services context. Workflows are generated semi-automatically with the integration of the Chimera system [13]. Splunter *et al.* [28] propose a fully automated agent-based mechanism for web service composition and execution using an open matching architecture. In a similar vein to these two approaches, our work aims to provide semi-automatic and automatic means for workflow composition, but does not deal with the mapping of resources onto the workflow components. In our approach, the task of management of workflow execution at run time is performed by a separate resource scheduler that will manage the distribution of the tasks.

 © Fish4Knowledge Consortium, 2012

4 Compute Environment

The compute environment for F4K is provided by NCHC in Taiwan. The environment comprised of a supercomputer, a PC cluster and several virtual machine (VM) servers. The supercomputer, Advanced Large-scale Parallel Supercluster (ALPS), named 'Windrider', has a total of 8 compute clusters, 1 large memory cluster, and over 25,000 compute cores. Out of these, 2 nodes containing 48 cores each (a total of 96 cores) were dedicated to F4K. It runs on Novell SuSE Linux Enterprise 11 SP1. Although it has the Load Sharing Facility (LSF) scheduler ready for use, we are still faced with several security firewalls with regards setting up our working environment. We anticipate to use these 96 cores for executing F4K tasks in the near future once preliminary testing is over.

The PC cluster, known as 'gad245' has one master node and four compute nodes. It operates on Ubuntu 11.10 kernel 3.0. Each node has 8 cores that can be used for computation. At present the HPC team (NCHC) is working on the deployment of a VM cluster system to replace the original PC cluster to ease further progress. Hence it is not ready for use yet.

Two VM servers are dedicated to F4K researchers, 'gad246' and 'gad247'. Both run on Ubuntu 11.10 kernel 3.0. VM gad246 is a 48-core machine with 24 cores dedicated to F4K. 8 cores are allocated for workflow use, and 16 cores for the two image processing teams (8 cores each). One of these 8-core VM is used for storing the central database. Gad247 is a 16-core machine with 12 cores dedicated for workflow use. This VM has been used as our main development environment. The workflow manager, scheduler and VIP modules have been installed on this VM and it will act as the master node for the workflow development and testing. Fig. 2 summarises the HPC facilities available for F4K's use.

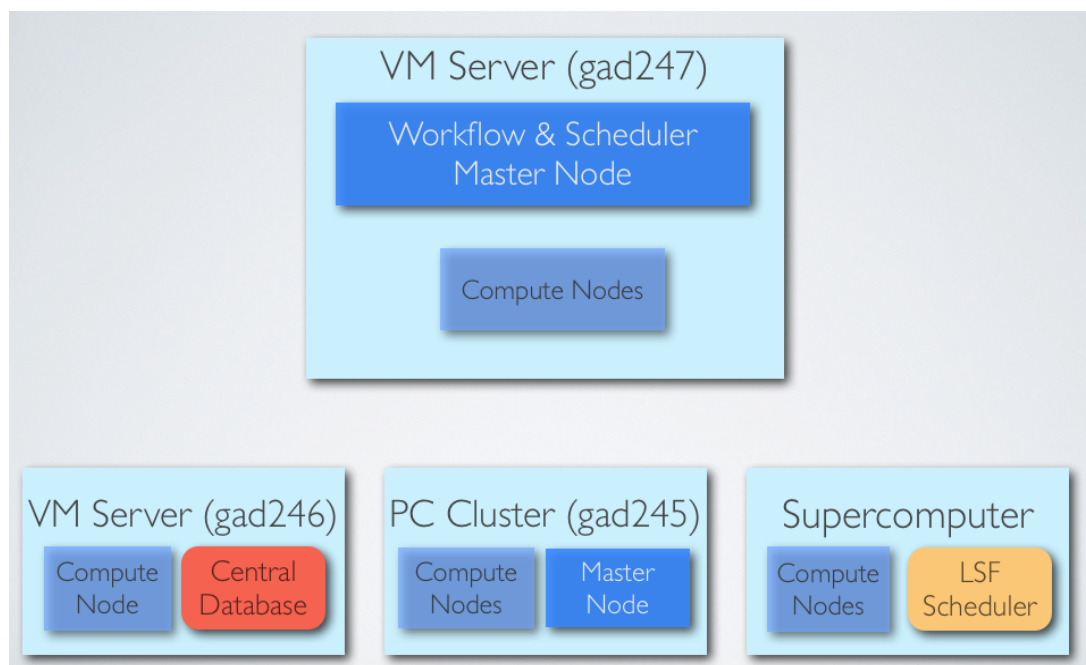

Figure 2: The compute environment available for F4K's workflow use. The VM server gad247 will act as the master and (10-core) compute node. VM gad246 will also act as a (8-core) compute node for preliminary workflow testing. The PC cluster is in the process of being transformed into a VM cluster. The supercomputer, Windrider, is still being investigated for future use due to security issues.

 © Fish4Knowledge Consortium, 2012

5 Workflow Design Framework

Based on the motivations and requirements outlined in the previous sections, a workflow framework was designed. This framework incorporates the knowledge available to the domain (captured in the goal, video description and capability ontologies [23]) and the compute environment (hardware resources) available. First the core functions of the workflow are identified (Section 5.1), next the architecture design of the workflow is presented (Section 5.2), highlighting its main components and functions. Then the workflow composition is explained (Section 5.3).

5.1 Workflow Functions

Discussions with the user interface team has lead to the identification of the core functions that the workflow should support. Some of these were not identified as major requirements for the workflow in Section 2, rather they were formulated discussions with the user interface team that will make use of the workflow functions. The roles of the workflow are as follows:

1. Perform **on-demand workflow queries** (high priority) from user interface - compose, schedule, execute and monitor jobs. On-demand queries are the most computationally intensive tasks that the workflow will have to perform, e.g. fish detection and tracking and fish species recognition. It is therefore crucial that latency is minimised for these types of tasks. The execution of these tasks will have to be monitored in order to report feedback to the user and also to handle exceptions.

2. Perform **batch/self-managed workflow queries** (low priority) on new unprocessed videos from NCHC's source - compose, schedule, execute and monitor jobs. These tasks are essentially the same type of tasks as on-demand queries but are not triggered by the user interface, instead by the workflow server API *e.g.* daily processing of new videos. These tasks are of low priority and can be run in the background.

3. Perform **run-time estimation** for a given workflow query when asked by the user interface. This involves the calculation of the time estimated for a query to execute. This would involve considering several factors, such as the number of videos required, the computational intensity of the VIP modules involved and the availability of HPC machines.

4. Update the database with the **progress of execution** for each on-demand worklfow query during short intervals of execution. Thus it can retrieve the progress of a workflow query's execution (as a percentage) when asked by the user interface.

5. **Stop the execution** of a task when asked by user (abort). This would stop the execution of a task, however results already stored in the database will not be removed.

6. **Report failure** of a task to user interface (beyond fault detection and repair). The workflow will deal with exceptions such as software and hardware failures using fault tolerant strategies. However, for drastic failures where such measures are not able to deal with the exceptions, a failure will be reported to the front end.

 © Fish4Knowledge Consortium, 2012

5.2 Workflow Manager Architecture

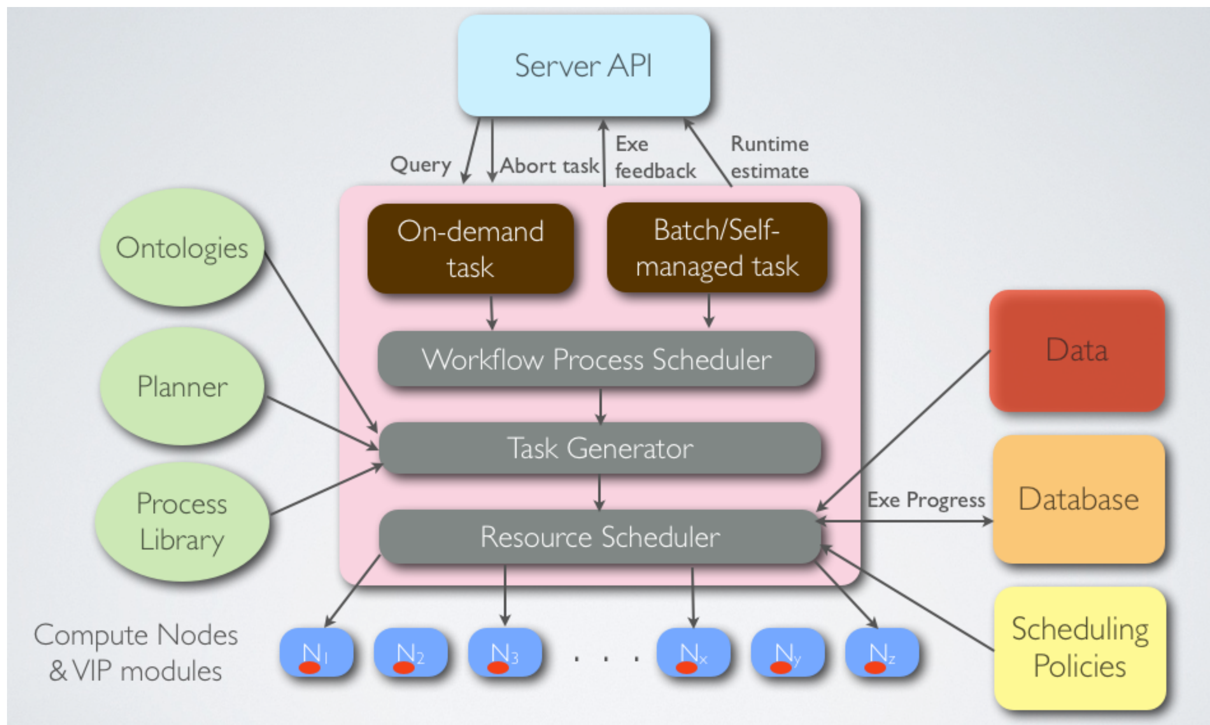

Figure 3: The workflow component binds high level queries from the user interface to low level image processing components via process planning and composition. It also schedules and monitors the execution of the video processing tasks on a high performance computing environment and reports feedback to the user interface component.

The workflow manager's architecture diagram (Fig. 3) shows an overview of the components that the workflow interacts with, its main functions, and its sub-components. It interacts with the front end via a server API and updates information in the central database. As can be seen there are three workflow management sub components: 1) Workflow Process Scheduler; 2) Task Generator and; 3) Resource Scheduler.

The Workflow Process Scheduler deals with high level process management, such as manipulation of queries. In particular, it deals with the modification of queries so that no overlaps are present. Overlaps can occur when multiple queries contain identical values in the task, dates, location and site. Multiple queries can be invoked from the front end which is a web-based user interface. For example, "Detect and track fish in Lanyu island cameras from 1st January 2011 to 30th June 2011" and "Detect and track fish in Lanyu island cameras from 1st March 2011 to 30th September 2011". The portion that overlaps are the processing between 1st March and 30th June 2011. The process scheduler ensures that this portion is not executed twice in order to save processing time for on-demand queries.

The process scheduler then deals with breaking down this high level query into individual VIP tasks that each act on a video clip. Fig. 4 shows the tasks generated for the query detect and track fish. It loops over all the videos (every 10 minutes) between the start and end dates and over all the cameras selected at each 10-minute span. For each video clip, a sequence of VIP operations are required for this task to be accomplished.

 © Fish4Knowledge Consortium, 2012

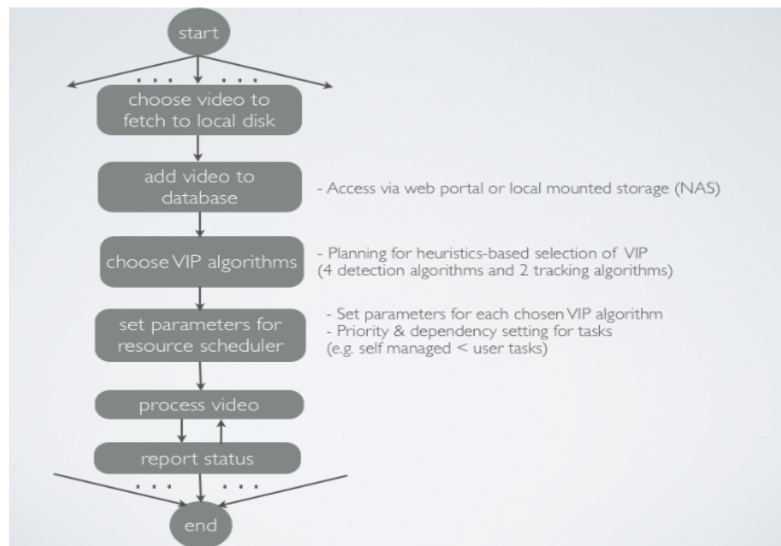

Figure 4: Example task generation for fish detection and tracking for each 10-minute video clip.

Once the portion of queries are ready for processing, they are passed to the Task Generator to be composed as video processing workflows. The Task Generator is the workflow composition engine which utilises planning and ontologies. This mechanism will be described in detail next. The progress of the integration of the workflow with the Resource Scheduler is provided in Section 6.4.

5.3 Workflow Composition using Planning

The workflow composition mechanism was devised based on a three-layered framework implemented in earlier versions of the workflow prototypes [21, 22]. Fig. 5 gives a pictorial overview of the workflow composition framework.

The **design layer** contains components that describe the domain knowledge and available video processing tools. These are represented using ontologies and a process library. Knowledge about image processing tools, user-defined goals and domain description is organised qualitatively and defined declaratively in this layer, allowing for versatility, rich representation and semantic interpretation. The ontologies which are used for this purpose are described in [23] and also contained in Deliverable 3.1. The process library developed in the design layer of the workflow framework contains the code for the image processing tools and methods available to the system. These are known as the process models. A set of primitive tasks are identified first for this purpose. A primitive task is one that is not further decomposable and may be performed directly by one or more image processing tools, for instance a function call to a module within an image processing library, an arithmetic, logical or assignment operation. Each primitive task may take in one or more input values and return one or more output values. Additionally, the process library contains the decomposition of non primitive tasks or *methods*. This will be explained in the next subsection.

The **workflow layer** is the main interface between the front end and the back end of the F4K system. It also acts as an intermediary between the design and processing layers. The Workflow Process Scheduler deals with higher level management of user interface-supplied

 © Fish4Knowledge Consortium, 2012

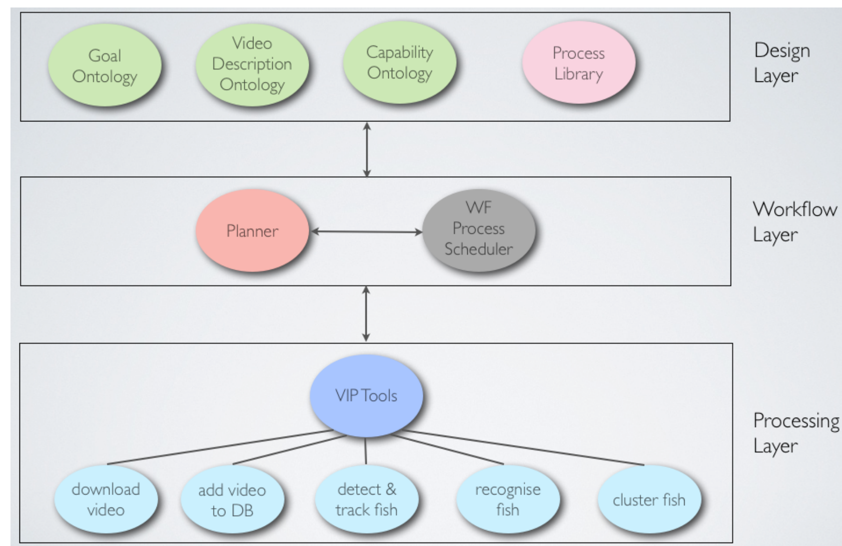

Figure 5: Overview of workflow composition framework for video processing. It provides three levels of abstraction through the design, workflow and processing layers. The core technologies include ontologies and a planner.

queries. The main reasoning component is an execution-enhanced planner that is responsible for transforming the high level user requests into low level video processing solutions. Detailed workings of the planner is contained in the next section.

The **processing layer** consists of a set of VIP tools that can perform various image processing functions. The tools developed so far are contained in Fig. 5. The functions of these tools are represented in the capability ontology in the design layer. Once a tool has been selected by the planner, it is put on a queue to be sent to the resource scheduler. When execution is complete, the database is updated with the results and the user interface is notified. The set of VIP tools available for performing various image processing operations are generated using OpenCV [17] and Matlab [19].

5.4 Hierarchical Task Network (HTN) Planning

The planner is a reasoner that translates the high level VIP goals (provided by the front end) to low level VIP operations for workflow composition. This is done with the assistance of the process library and ontologies. The planner was built based on ordered task decomposition where the planning algorithm which composes tasks in the same order that they will be executed. This class of planners are known as Hierarchical Task Network (HTN) planners. The input to the planner are the goals, objects and conditions of the objects at the beginning of the problem (initial state), and a representation of the actions that can be applied directly to primitive tasks (operators). For HTN planning, a set of methods that describe how non primitive tasks are decomposed into primitive and non primitive tasks are also required. The output should be (partial) orderings of operators guaranteed to achieve the goals when applied to the initial state.

A video processing task can be modelled as an HTN planning problem, where a goal list, G is represented as the VIP task(s) to be solved, the primitive tasks, p are represented by the VIP primitive tasks and the operators, O are represented by the VIP tools that may perform the

 © Fish4Knowledge Consortium, 2012

primitive tasks directly. The methods, M specify how the non primitive tasks are decomposed into primitive and non primitive subtasks. The primitive tasks and methods are contained in the process library.

5.4.1 Preliminaries

Adapting the conventions provided in Ghallab *et al.* [14], an HTN planning problem is a 5-tuple

$$P = (s_0, G, P, O, M)$$

where s_0 is the initial state, G is the goal list, P is a set of primitive tasks, O is a set of operators, and M is a set of HTN methods. A primitive task, $p \in P$ is a 3-tuple

$$p = (\text{name}(p), \text{preconditions}(p), \text{postconditions}(p))$$

where name(p) is a unique name for the primitive task, preconditions(p) is a set of literals that must hold for the task to be applied and postconditions(p) is a set of literals that must hold after the task is applied.

An HTN method, $m \in M$ is a 6-tuple

$$m = (\text{name}(m), \text{task}(m), \text{preconditions}(m), \text{decomposition}(m), \text{effects}(m), \text{postconditions}(m))$$

where name(m) is a unique name for the method, task(m) is a non primitive task, preconditions(m) is a set of literals that must hold for the task to be applied, decomposition(m) is a set of primitive or non primitive tasks that m can be decomposed into, effects(m) is a set of literals to be asserted after the method is applied and postconditions(m) is a set of literals that must hold after the task is applied. The planning domain, D, is the pair (O, M).

Using the declarative approach in Prolog, the current state of the world is represented by predicates that hold (true). When the state of the world changes, for example, the timestamp has increased or a subgoal has been achieved, predicates are retracted (removed) and asserted (added) with appropriate values to reflect this.

5.4.2 Primitive Tasks, Operators and Methods

Five VIP operators were implemented for performing core video processing functions in F4K so far. They are i) downloading a video to the local disk, ii) adding a video on local disk to the database, iii) performing detection and tracking tasks on a video, iv) performing fish species recognition on detected fishes and v) performing fish clustering on detected fishes.

The corresponding primitive tasks that the operators can act upon are encoded in the process library. As stated earlier, primitive tasks are those that can be performed directly by operators or VIP tools. For each primitive task, its corresponding technical name, preconditions, parameter values, output values, postconditions and effects of applying this task are specified. The preconditions are all the conditions that must hold (prerequisites) for this primitive task to be performed. The effects are conditions that will be asserted or retracted after completion of the task and the postconditions are all the conditions that must hold *after* the task is applied. Appendix 8.1 contains the contents of the process library developed for the F4K workflow.

Non primitive tasks are decomposable to primitive and non primitive subtasks. Schemes for reducing them are encoded as *methods* in the process library. For each method, the name of the method, the preconditions, decomposition, effects and postconditions are specified. The

 © Fish4Knowledge Consortium, 2012

decomposition is given by a set of subtasks that must be performed in order to achieve this non primitive task. For VIP tasks, the methods are broadly categorised into three distinct types; non recursive, recursive and multiple conditions. Non recursive method is the most common form that does not involve loops or branching of any sort, for example a direct decomposition of a video classification method into processing the video frames, followed by performing the classification on the frames and finally writing the resulting frames onto a video. Recursive methods, as its name suggest, model loops, hence the decomposition of these methods will include itself as a subtask. The latest version of the workflow does not contain recursive methods although previous versions did. This modification eases distributed execution that would be problematic with recursions. Multiple conditions arise when there is more than one decomposition for a method. For instance, there are two ways to download a video, one is through a web download (more conventional format) and the other is via accessing the mounted storage device. Two separate decompositions will be provided for the method download video. The choice of selecting which method is preferred can be done via the ordering of the methods; the one which is stated first will be selected first, failing that, the second method will be selected, or by specifying the preconditions that must hold for that method to be selected.

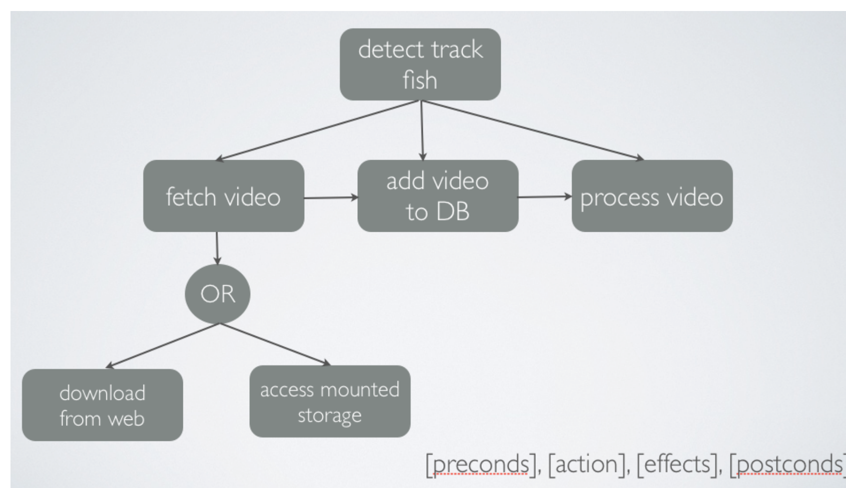

Figure 6: Hierarchical planning for fish detection and tracking task.

5.4.3 Planning Algorithm

Planning algorithms solve problems by applying *applicable* operators to initial state to create new states, then repeating this process from the new states until the goal state is reached. In HTN planning, the algorithm stops when all the goal tasks have been achieved. The algorithm below describes the main workings of the planner implemented for F4K's workflow.

```
gplanner(initial-state S, goal-list [g1|G], domain D, solution-plan P)
Initialise P to be empty
If goal-list is empty, return P

Consider first element in the goal-list, g1 in goal list [g1|T]
Case 1: If g1 is primitive AND all its preconditions hold
       1.1. If one or more operator instances (tools) match g1
            Retrieve the tool tp from capability ontology that can perform g1
            tp = check_capability(g1)
                Apply tp to S and planning algorithm to rest of goal-list:
```

 © Fish4Knowledge Consortium, 2012

```
                apply-operator(tp, Input_list, Add_list)
                Check that all postconditions of g1 hold
                gplanner(tp(S), G, D, P)
       1.2 Else return fail

Case 2: If g1 is non primitive
       2.1. If a method instance m matches g1 in S
            AND all its preconditions hold
              Append the decompositions of m into the front of the goal-list
              Add all elements in m's Add List to S
              Check that all postconditions of m hold
              Apply planning algorithm to this new list of goals:
              gplanner(S, append(m(g1),G), D, P)
       2.2 Else return fail

% check_capability
check_capability(planning-step g)
  Retrieve a tool, T that can perform g from capability ontology:
  getTool(g)
  Check that domain-criteria tied to this tool (if any) hold:
  check_criteria(T)
return T

% getTool
getTool(S)
  canPerform(T, S)
  instance(T, T1)
  descendant_of(T1, tool)
Return T

% check_criteria
check_criteria(T)
  If a set of domain-criteria, DC exist for this tool
  hasPerformanceIndicator(T, DC)
    Retrieve the list of preconditions, P for DC:
    instance_att_list(DC, P)
      If all preconditions in P hold
         return DC
      Else return Fail
  Else return no_criteria

% Apply_operator
apply-operator(Tool, Input_list, Add_list, P, S)
    Update solution-plan P with Tool (append Tool to the end of P)
    Put Tool with parameters Input_list on queue for resource scheduler
    Add all elements in Add_list (effects) to S
```

Algorithm 1: The workflow planner's algorithm using HTN planning with ontologies.

As explained in Section 5.4.1, the domain or current state of the world reflects whatever facts or predicates that hold at a particular time. The domain is represented by the predicates that contain the goal, timestamps, the constraints (*e.g.* accuracy and processing time), methods (decompositions) and operators (tools to execute the primitive tasks). The algorithm is a recursive function that works on the goal list until it is empty. It inspects each item in the goal list to see if it is a primitive task. If the item is a primitive task, it seeks to find an operator that can perform the primitive task. Once found, the operator is applied and the primitive task is accomplished. If the task is not primitive, it looks for a method instance that matches it and appends its decompositions to the start of the goal list. The basis for the planning algorithm was taken from HTN planners that generate plans in a totally ordered fashion, *e.g.* SHOP [24]. Tasks are decomposed from left to right in the same order that they will be executed. In addition, it can plan interactively, interleave planning with workflow execution and has been enriched to make use of knowledge from ontologies.

 © Fish4Knowledge Consortium, 2012

6 Implementation

A baseline workflow system was implemented with the ability to compose workflows for core VIP tasks stated in Section 5.4.2. It is also able to perform a subset of the core functions stated in Section 5.1. The workflow was written using SWI-Prolog [30] version 5.10.4. In the remaining sections, the baseline workflow's implementation and technical details are provided. Explanations include invocation mechanism (Section 6.1), integration with VIP modules (Section 6.2), integration with the database (Section 6.3) and integration with the scheduler (Section 6.4).

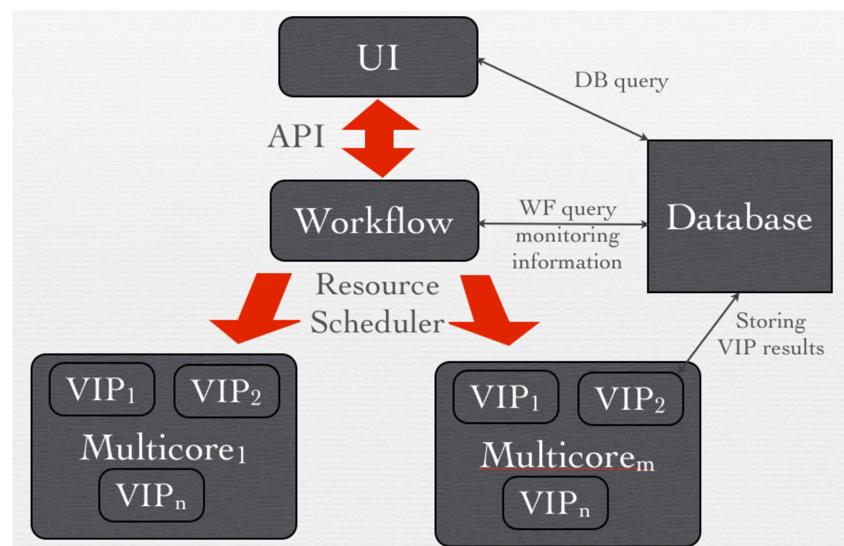

Figure 7: Sketch for workflow integration.

As can be seen in Fig. 7, the workflow component is connected to the front end user interface via a server API. It is connected to the computing machines via a resource scheduler. The VIP modules are installed in all the machines which will act as compute nodes. The central database is accessed by all the components and plays a key part in communication.

6.1 Invocation

The workflow supports five types of functions; execution of on-demand and batch workflow queries, run-time estimation for a VIP task, computation of the execution progress of a query, abortion of a query and notification of a query's failure. The workflow is invoked by a server API using command line arguments.

There are two modes in which the workflow can be invoked, basic and advanced. Basic invocation takes the parameters that must be supplied by the user interface. For the execution query, the format for invocation is as follows:

```
./workflow.pl <goal> <start-date> <end-date> <camera-list>
```

where <goal> is one of [detect_track_fish, cluster_fish, recognise_fish].

An example invocation for fish detection and tracking task in the Nuclear Power Plant (NPP-3) site (which has three cameras) in January 2010 is:

```
./workflow.pl detect_track_fish 2010 01 01 2010 01 31 'NPP-3/1, NPP-3/2, NPP-3/3'
```

 © Fish4Knowledge Consortium, 2012

For run-time estimation queries the format for invocation is as follows:

```
./workflow.pl runtime_estimate <goal> <start-date> <end-date> <camera-list>
```

where <goal> is one of [detect_track_fish, cluster_fish, recognise_fish] as above.
An example invocation of this workflow function is:

```
./workflow.pl runtime_estimate detect_track_fish 2010 01 01 2010 01 31 'NPP-3/1'
```

Finally for aborting task the invocation format is given below:

```
./workflow.pl abort <query_id>
```

where <query_id> is the unique identifier for that query that is generated by the workflow upon receiving the query from the user interface. This information is stored in the database and used by the workflow to monitor the queries.

The advanced workflow invocation allows additional parameters to be included with the basic call. These are appended to the end of the basic command line invocation. This is where user provided parameters, such as specific algorithm selections can be included. For example, to include the adaptive Gaussian mixture model detection algorithm and covariance tracking algorithm the following command line invocation is used:

```
./workflow.pl detect_track_fish 2010 01 01 2010 01 31 'NPP-3/1' detection=
agmm tracking=cov
```

6.2 Integration with VIP Modules

As mentioned in Section 5.1, the VIP modules that have been developed so far are as follows:

- Download a video from NCHC's website and save on the local disk.
- Add a video on the local disk to the database (as an entry in the *videos* table).
- Perform fish detection and tracking on an existing video and store the results in the database.
- Perform fish species recognition on an existing video with detected fish and store the results in the database.
- Perform fish clustering on an existing video with detected fish and store the results in the database.

These are some the VIP tasks that have been identified in Deliverable 3.1 as the main VIP goals and contained in the goal ontology. As development continues, more VIP modules will become available and added to the workflow's process library and capability ontology.

The first three modules were developed using OpenCV version 2.2 while the remaining two were developed using Matlab version 2009b. They were packaged as executables and invoked via command line arguments. Generally speaking, their invocations have the following formats:

1. <component> <camera-id> <date> [<video-quality-parameters> <video-name> <database-configuration>]
2. <component> <flag$_1$ > <value$_1$ > ... <flag$_n$ > <value$_n$ > <additional parameters>

 © Fish4Knowledge Consortium, 2012

3. <component> <dependent-files> <database-name> <video-id> <algorithms> <data-store-dir-and-remove-options>

4. <component> <runtime-environment> <model-file> <readfrom-database> <writeto-database> <video-id> <component-id> <segmentation-method> <data-store-dir-and-remove-options>

In each case, the workflow is able to extract the date and camera from the input given. It can also extract the video-id from the date and timestamp values in the database. The values that workflow will have to determine are the detection, tracking and segmentation algorithms. Efforts are underway to standardise and modularise the way the VIP modules' parameters are designed and invoked. Based on discussions with IP teams, the draft specification in Table 1 was produced and will continue to evolve throughout the development phase.

Table 1: Draft specification for VIP modules' development and deployment.

Item	Value(s)	Status
Flags	Numerical/Preceding String e.g. 0/1, –db, -v	To be decided
Database settings	Configuration file/String/None	Use configuration file
Documentation	Readme file	Description of parameters & allowed values and a sample default run should be included
Directory (module)	/home/gaya/components/<goal>	Absolute paths should be used throughout. <goal> is one of detect_track_fish, fish_recognition, fish_clustering, download_video, add_video_to_db for now
Directory (data)	home/gaya/tmp/video_dir	Absolute paths should be used throughout
Directory (results)	/home/gaya/tmp	Absolute paths should be used throughout
Results	Stored/Removed	To be decided
Dependencies	Yes (module name)/No	Does it assume results from other modules?
Fault tolerance & Messages	Log error in DB and logfile /components/video_id.err	
Side Effects		Should not cause database (or other) errors if rerun with same parameters

Outstanding issues include how to streamline errors caught by the VIP modules with the errors caught by the workflow. It should be noted that software or machine failure will be caught by the workflow, however, errors that can be caught by the VIP modules should be communicated to the workflow.

6.3 Integration with Database

F4K has opted for MySQL [20] database for storing VIP results, workflow query monitoring progress and communication information. It is hosted in a server in Uni. Catania with the intent of full migration to the Taiwanese VM as soon as possible. The workflow uses ODBC to connect to the F4K database. The main database tasks that involve the workflow are monitoring high level queries and low level VIP tasks (jobs) that are submitted via a resource scheduler. The workflow also retrieves video ids for processing of tasks and updates the *videos* table

 © Fish4Knowledge Consortium, 2012

when a video is added to the database. The high level queries are monitored using the table *query_monitoring* described in Fig. 8 below.

	Field	Type	Collation	Attributes	Null	Default	Extra
☐	query_id	int(11)			No	*None*	auto_increment
☐	wf_process_id	int(11)			No	*None*	
☐	query_task_name	varchar(100)	latin1_swedish_ci		Yes	*NULL*	
☐	start_date	date			Yes	*NULL*	
☐	end_date	date			Yes	*NULL*	
☐	status	varchar(255)	latin1_swedish_ci		Yes	*NULL*	
☐	percentage_complete	int(11)			Yes	0	
☐	current_timestamp	timestamp		on update CURRENT_TIMESTAMP	No	CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP

Figure 8: Database schema for the *query_monitoring* table.

Each query has a unique query id, a task name associated with it (the VIP goal), start date and end date. It also contains a status field to indicate if the query is 'Queued', 'Executing', 'Completed', 'Failed' or 'Aborted'. The *percentage_complete* field is used to monitor the portion of the query has been executed. This will be an integer value (between 0 and 100) to indicate the percentage of completed runs. The *currect_timestamp* field keeps track of when the table entry was last modified.

As explained in Section 5.2 the workflow will deal with the high level manipulation of breaking down a query into multiple VIP tasks or jobs. Each individual VIP module execution willl also need to be monitored, in a similar fashion to the queries. The monitoring details are logged into the table *processed_videos* in the database which contains the status, progress and timestamp values. At present the workflow does not update the *processed_videos* table, as it is done by the corresponding VIP module during execution. However, once the workflow is integrated with the scheduler, it will be able to manipulate this table as well. Section 6.5 will highlight the database implications of the different queries dealt by the workflow.

6.4 Integration with Scheduler

A resource scheduler that can deal with allocating the workflow jobs onto multiple machines was sought. A candidate scheduler that stood out from our research is the Oracle (formerly Sun) Grid Engine (SGE) [8]. The gridegine version that was bundled with the development environment (Ubuntu 11.10) was installed and set up for testing. For the setting up of this scheduler, a master node was elected (gad 247). The master node also contains the workflow manager where the jobs will be submitted to the scheduler from. The requirements for the installation and setup of SGE master and compute nodes on our working environment include:

1. Install gridengine server packages on the master node.
2. Need ssh access to all compute nodes from the master node without password.
3. Nodes need to be able to do forward and backward resolution on their domain names.
4. Install gridengine-exec and gridengine-client packages on all compute nodes.
5. May need a shared (NFS) filesystem in order to get full job data back from the nodes. Either the master node or a dedicated server is nominated as the fileserver.

 © Fish4Knowledge Consortium, 2012

As for invoking the scheduler, the workflow automatically generates command line calls in the following format:

```
qsub <options> <command>
```

where qsub submits a batch job to the SGE queuing system. SGE supports single- and multiple-node jobs. Command can be a path to a binary or a script which contains the commands to be run by the job using a shell. In our case the command will correspond to the invocation of one of the VIP modules described in Section 6.2. For example, to queue a (basic) download video job, the following command is generated and invoked by the workflow:

```
qsub /home/gaya/components/detect_track_fish/download_video.sh NPP-3
1 201106030800 1 '1 2' 2 '5 8 24'
Your job 20 ("download_video.sh") has been submitted
```

Monitoring and configuring the queued jobs are done via the 'qstat' and 'qacct' commands. Options can be requested either hard or soft. By default, all requests are considered hard until the -soft option is encountered. The hard/soft status remains in effect until its counterpart is encountered again. If all the hard requests for a job cannot be met, the job will not be scheduled. Jobs which cannot be run at the present time remain spooled. The list of commands available for invoking SGE can be found in the Gird Engine manual pages [7].

Using the SGE scheduler, the workflow can take advantage of several configuration options that can be set prior to sending jobs for execution. At present the scheduler's capabilities that are relevant for the workflow are summarised as below:

- By default submitting a job will schedule it in queue style (FIFO).
- Can set queues by priority.
- Can set jobs based on load-balancing options.
- Can set job dependencies.
- Can set the number of cores (CPUs) for a job or list of jobs.

We anticipate to evaluate the effectiveness of setting priorities to queues (on-demand and batch jobs) and running them simultaneously on nominated number of CPUs once full integration with the scheduler is in place. Currently the scheduler is being set up in gad247 and gad201 and simple testing will proceed when setup is complete. The next section provides some sample runs of the baseline workflow prototype.

6.5 Sample Runs

In this section a few sample runs of the workflow calls are provided. This is to demonstrate the extent of the baseline workflow capabilities.

 © Fish4Knowledge Consortium, 2012

6.5.1 Execution Queries

Three types of execution queries are possible at the moment: i) fish detection and tracking; ii) fish species recognition (on the detected fish); and iii) fish clustering (on the detected and recognised fish). When such a query is posed to the workflow, it follows a series of steps to manage its execution:

1. First, an entry in the *query_monitoring* table is created with its *query_id, wf_process_id, start_date, end_date, status* and *current timestamp*. The *query_id* is automatically generated while the *wf_process_id* is the process id of the workflow execution program itself. Its *status* is set to be 'Queued' and then changed to 'Executing' when jobs are invoked for execution. *Percentage_complete* is set to 0 by default.

2. Then, at every interval of 5 % of videos, its percentage of completion is calculated and updated in the *percentage_complete* field.

3. Finally, when all the videos have been processed, its status is updated to 'Completed' and percentage complete is set to 100 (provided it was not aborted or failed during execution).

An example fish detection and tracking task was called using the following workflow invocation:

```
> ./workflow.pl detect_track_fish 2011 06 01 2011 06 07 'NPP-3/1'
```

For this task, VIP workflow instances are generated and sent for execution as jobs. Sample workflow instances for <u>each video</u> are as follows:

```
$ /home/gaya/components/detect_track_fish/download_video.sh NPP-3 1
201106030800 1 '1 2' 2 '5 8 24'

$ /home/gaya/components/detect_track_fish/add_camera_video_to_db NPP-3 1
2011 06 03 08 00 /home/gaya/tmp/video_dir/NPP-3-1-2011-06-03-08-00.flv
/home/gaya/components/detect_track_fish/db_config.xml 2> /dev/null

$ /home/gaya/components/detect_track_fish/process_video -v /home/gaya/tmp/
video_dir/NPP-3-1-2011-06-03-08-00.flv --tracking-eval-model /home/gaya/
components/detect_track_fish/tracking_eval_model.txt -i 9825 agmm cov --db
```

For each query, these instances are stored in the script file /home/gaya/workflow/wf-scripts/<query_id>.sh for reference and can be invoked offline. The workflow communicates to the front end via database updates, messages to standard output and log files. The directory /home/gaya/workflow/logs contains log files to store the progress of execution and error logs. The progress of execution is stored in file called <query_id>.log and the error logs are contained in the file <query_id>.err. Th log and errors files of the execution of individual videos will be synchronised with the respective VIP components.

 © Fish4Knowledge Consortium, 2012

6.5.2 Runtime Estimate

The aim of this task is to give an estimated waiting time for the user should they want to pose a query. At present the workflow handles this task in a very naive way. It estimates five minutes' computation per video, calculated over the total number of video clips for that query (with an assumption of only 12 hours worth of videos per day) and run over 5 processors which run 5 jobs each. Runtime estimation is a complex problem in itself and would require further investigation, such as the total number of processors used, the computational intensity of the VIP module and the network speed. A sample run and result of a fish detection and tracking query in one site over a week is given below.

```
$ ./workflow.pl runtime_estimate detect_track_fish 2011 06 01 2011 06 07 'NPP-3/1'
% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec, 3,856 bytes
%     library(error) compiled into error 0.00 sec, 18,280 bytes
%    library(lists) compiled into lists 0.00 sec, 45,000 bytes
%   library(shlib) compiled into shlib 0.00 sec, 66,216 bytes
%  library(unix) compiled into unix 0.00 sec, 74,144 bytes
%  processLibrary-hpc.pl compiled 0.00 sec, 13,128 bytes
%  gplan-hpc.pl compiled 0.00 sec, 9,728 bytes
% /home/gaya/workflow/workflow.pl compiled 0.01 sec, 169,760 bytes
Goal is runtime_estimate
Num clips 504.0 Num_cameras 1
Estimated runtime for task runtime_estimate with 504.0 clips is 50.400 minutes
```

6.5.3 Progress of Execution

This query is essentially a quick lookup in the *percentage_complete* field in the *query_monitoring* table. However, the values contained in this field is frequently updated by the workflow manager. At present, this is done in intervals of 5% of videos. Each day has approximately 72 videos (12 hours). A sample run for the progress of execution command is as follows:

```
$ ./workflow.pl execution_progress 29
% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec, 3,856 bytes
%     library(error) compiled into error 0.00 sec, 18,280 bytes
%    library(lists) compiled into lists 0.00 sec, 45,000 bytes
%   library(shlib) compiled into shlib 0.00 sec, 66,216 bytes
%  library(unix) compiled into unix 0.00 sec, 74,144 bytes
%  processLibrary-hpc.pl compiled 0.01 sec, 13,200 bytes
%  gplan-hpc.pl compiled 0.00 sec, 9,608 bytes
% /home/gaya/workflow/workflow.pl compiled 0.01 sec, 173,008 bytes
Goal is execution_progress
Query_id 29
f4k_db connected
Percentage completed: 40
f4k_db disconnected


******************* Completed ***********************
*
* Finished task execution_progress in 2.437 seconds *
*
*****************************************************
```

 © Fish4Knowledge Consortium, 2012

query_id ▲	wf_process_id	query_task_name	start_date	end_date	status	percentage_complete	current_timestamp
10	28220	detect_track_fish	2011-06-03	2011-06-03	Queued	0	2012-05-31 14:43:52
11	28305	detect_track_fish	2011-06-03	2011-06-03	Aborted	0	2012-05-31 14:45:45
17	29319	cluster_fish	2011-06-03	2011-06-03	Executing	0	2012-05-31 20:03:45
29	31874	detect_track_fish	2011-06-03	2011-06-03	Executing	80	2012-05-31 23:58:14
34	4153	detect_track_fish	2011-06-05	2011-06-05	Completed	100	2012-06-01 16:44:19
35	5273	detect_track_fish	2011-06-06	2011-06-06	Executing	67	2012-06-01 17:12:03

Figure 9: Table *query_monitoring* showing different progress of execution levels contained in the *precentage_complete* field and different statuses of queries.

6.5.4 Abort Query

Aborting an executing task requires two actions: i) stopping the job(s) spawned for the query in question and ii) updating the database entry for this query as 'Aborted'. At present the workflow job that runs the query is kept track in *query_monitoring* table via the *wf_process_id* field. A sample run is shown below:

```
gaya@gad202:~/workflow$ ./workflow.pl abort 11
%     library(error) compiled into error 0.00 sec, 18,280 bytes
%    library(lists) compiled into lists 0.00 sec, 44,808 bytes
%   library(shlib) compiled into shlib 0.00 sec, 66,024 bytes
%  library(unix) compiled into unix 0.00 sec, 73,952 bytes
%  processLibrary-hpc.pl compiled 0.00 sec, 12,376 bytes
%  gplan-hpc.pl compiled 0.00 sec, 10,536 bytes
% /home/gaya/workflow/workflow.pl compiled 0.01 sec, 169,880 bytes
Goal is abort
Aborting query 11
f4k_db connected
Process aborted: 28305
f4k_db disconnected


******************** Completed ************************
*
* Finished task abort in 2.787 seconds *
*
*******************************************************
```

The database entry for this query is updated. For example it can be seen in Fig. 9 that the status for query 11 is 'Aborted'. Using this method, only the process id of the main executing job (of the workflow) will need to be stopped. This will immediately stop each of its child processes that execute on individual videos. This mechanism of stopping the query's process id will change when integrated with the scheduler. Mechanisms to retrieve the scheduler job id will need to be in place.

 © Fish4Knowledge Consortium, 2012

7 Discussion and Summary

The baseline workflow is a crucial step towards achieving integration in the back end of the F4K system. At present, the baseline workflow is able to run the specifed workflow tasks with default values. It can connect to the database, invoke the VIP modules and queue simple jobs on the scheduler. It is not integrated with the user interface yet. In terms of dealing with high level queries, it can break tasks down according to the dates and camera list to invoke VIP modules. It is also able to select default values for algorithms in the five VIP modules supplied to it. It is able to create database entries for new videos, new queries and abort queries. In addition, it is able to update the progress of execution for a query. Runtime-estimation is only naive at the moment and will need further investigation.

Development will continue, especially in getting full integration and manipulation of scheduler configurations and data controller status. Rigorous testing will bring to light the technical barriers that will need to be overcome for full integration. Error handling is another focus for the next development stage of the workflow. Issues that continue to persist include missing data (due to disruption of video recording, power cuts, facilities failures, etc.), corrupted data (unusable), heterogeneous working environments, and database technicalities.

 © Fish4Knowledge Consortium, 2012

8 Appendix

8.1 Process Library

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Process library for intelligent workflow                  %%
%% Gayathri Nadarajan                                        %%
%% 30/05/12                                                  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% method(Name, Precond, Decomp, Effects, Postcond).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
method(detect_track_fish, [], [fetch_video, process_video], [], []).
method(fetch_video, [], [download_video, add_camera_video_to_db], (nl), []).
method(fetch_video, [], [read_from_mounted_storage, add_camera_video_to_db], (nl), []).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 0. Primitive tasks                                        %%
%%                                                           %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% independent_executable(User_term, Fn_call, List_Preconds, Add_list, Input, Output, List_Postconds).

primitive(PP, Fn_call, Input, Preconds, Add_list_file, Postconds) :-
    independent_executable(PP, Fn_call, Preconds, Add_list_file, Input, _Output, Postconds).
        %write('Independent executable '), write(PP), write(' found '),nl.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% independent_executable(User_terminology, Fn_call, Preconds_list,
%% Assert_list_file, Input_list, Output_list, Postconds_list)
%% method(name, precond, transition, statics, temps, decomposition).
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% gaya 28/3/12 fetch_video options
independent_executable(download_and_add_to_db, 'download_and_add_to_db.sh', [], [], [Site, CID,
Year, Month, Day, HourMin, Dir], [], []) :-
        get_site_cid(Site, CID),
        get_date_details(Year, Month, Day, _Hour, _Min),
        get_hour_min(HourMin),
        writeln(HourMin),
        get_video_directory(Dir).

independent_executable(download_video, 'download_video.sh', [], [], [Site, CID, Date, Res_Choice,
Res, 2, Fps_array]
, [], []) :-
        get_date(Date),
        get_site_cid(Site, CID),
        Res_Choice is 1,
        Res = '\'1 2\'',
        Fps_array = '\'5 8 24\''.

independent_executable(add_camera_video_to_db, add_camera_video_to_db, [], [], [Site, CID, Year,
Month, Day, Hour, Min, Vid_path_name, DB_config, Dev_null], [], []) :-
        get_date_details(Year, Month, Day, Hour, Min),
        Dev_null = '2> /dev/null',
        get_site_cid(Site, CID),
        get_video_name(V_name),
        get_video_directory(V_dir),        atom_concat(V_dir, V_name, Vid_path_name),
        goal(G), get_component_directory(G, C_dir),
        atom_concat(C_dir, 'db_config.xml', DB_config).

independent_executable(process_video, process_video, [video_exists(Video_path_name)], [], ['-v',
Video_path_name, '--tracking-eval-model', Track_eval, '-i', Video_id, Detect_algo, Track_algo,
'--db'], [], []) :-
        get_date_details(Year, Month, Day, Hour, Min),
```

```
        get_site_cid(Site, CID),
        get_video_name(V_name),
        get_video_directory(V_dir),
        atom_concat(V_dir, V_name, Video_path_name),
        goal(G), get_component_directory(G, C_dir),
        atom_concat(C_dir, 'tracking_eval_model.txt', Track_eval),
        get_video_id(Video_id),
        Detect_algo = 'agmm',
        Track_algo = 'cov'.

independent_executable(cluster_fish, 'run_fish_cluster_component.sh', [], [], [Matlab_compiler,
Model_file, DB_read_from, DB_write_to, Video_id, Component_id, Seg_method, V_store_dir,
Flag_remove_videos, Fish_store_dir, Flag_remove_fish], [], []) :-
        Matlab_compiler = '/opt/MATLAB/MCR/v715/',
        goal(G), get_component_directory(G, C_dir),
        atom_concat(C_dir, 'EmptyIndexingFileFishImagesDistv4.mat', Model_file),
        DB_read_from = 'mysql_catania',
        DB_write_to = 'mysql_catania',
        get_video_id(Video_id),
        Component_id is 16,
        Seg_method is 1,
        V_store_dir = '~/tmp/',
        Flag_remove_videos is 1,
        Fish_store_dir = '~/images/',
        Flag_remove_fish is 1.

independent_executable(recognise_fish, 'run_fish_recognition_component.sh', [], [], [Matlab_
compiler, Model_file, DB_read_from, DB_write_to, Video_id, Component_id, Seg_method,
V_store_dir, Flag_remove_videos], [], []) :-
        Matlab_compiler = '/opt/MATLAB/MCR/v715/',
        goal(G), get_component_directory(G, C_dir),
        atom_concat(C_dir, 'data_default.mat', Model_file),
        DB_read_from = 'mysql_catania',
        DB_write_to = 'mysql_catania',
        get_video_id(Video_id),
        Component_id is 16,
        Seg_method is 1,
        V_store_dir = '~/tmp/',
        Flag_remove_videos is 1.
```

References

[1] R. Allen. Workflow: An Introduction. In Layna Fisher, editor, *Workflow Handbook*, chapter 1, pages 15–38. Future Strategies Inc., 2001.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *Business Process Execution Language for Web Services Version 1.1 (BPEL)*. IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems. http://www-128.ibm.com/developerworks/library/specification/ws-bpel. Last accessed: Apr 21st, 2012.

[3] J. Bang-Jensen and G. Z. Gutin. *Digraphs: Theory, Algorithms and Applications, 2nd edition*. Springer Publishing Company, Inc., 2008.

[4] J. Blythe, E. Deelman, and Y. Gil. Automatically Composed Workflows for Grid Environments. *IEEE Intelligent Systems*, 19(4):16–23, 2004.

[5] G. Booch, I. Jacobson, and J. Rumbaugh. *OMG Unified Modeling Language Specification*. Version 1.3 First Edition, 2000.

 © Fish4Knowledge Consortium, 2012

[6] Y. H. Chen-Burger and F. P. Lin. A Semantic-based Workflow Choreography for Integrated Sensing and Processing. In *The 9th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'05)*, 2005.

[7] Oracle Corporation. *Grid Engine Man Pages*. 2012. `http://gridscheduler.sourceforge.net/htmlman/manuals.html`. Last accessed: May 25th, 2012.

[8] Oracle Corporation. *Open Grid Scheduler*. 2012. `http://gridscheduler.sourceforge.net/`. Last accessed: May 25th, 2012.

[9] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An Overview of Workflow System Features and Capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[10] E. Deelman, G. Singh, M.H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, K. Vahi, B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming Journal*, 13(3):219–237, 2005.

[11] Ecogrid. National Center for High Performance Computing (NCHC), Hsin-Chu, Taiwan. `http://ecogrid.nchc.org.tw`. Last accessed: Apr 21st, 2012.

[12] I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2nd edition, 2003.

[13] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In *14th Conference on Scientific and Statistical Database Management*, pages 37–46, 2002.

[14] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., 2004.

[15] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, and J. Kim. Wings for Pegasus: Creating Large-scale Scientific Applications using Semantic Representations of Computational Workflows. In *Proceedings of the 19th National Conference on Innovative Applications of Artificial Intelligence (IAAI'07)*, pages 1767–1774. AAAI Press, 2007.

[16] C. Girault and R. Valk. *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*. Springer-Verlag New York, Inc., 2001. `http://www.petrinets.info`. Last accessed: May 3rd, 2012.

[17] Intel. *Open Source Computer Vision (OpenCV) Library*. 2006. `http://sourceforge.net/projects/opencvlibrary`. Last accessed: Apr 21st, 2012.

[18] J. Kim, M. Spraragen, and Y. Gil. An Intelligent Assistant for Interactive Workflow Composition. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI'04)*, pages 125–131. ACM Press, 2004.

[19] Mathworks. *MATLAB - The Language of Technical Computing*. The MathWorks Inc., 1994-2012. `http://www.mathworks.com/products/matlab`. Last accessed: Apr 21st, 2012.

 © Fish4Knowledge Consortium, 2012

[20] MySQL. Oracle Corporation, 2012. http://www.mysql.com/. Last accessed: May 31st, 2012.

[21] G. Nadarajan. *Semantics and Planning based Workflow Composition for Video Processing*. PhD thesis, University of Edinburgh, 2010.

[22] G. Nadarajan, Y. H. Chen-Burger, and R. B. Fisher. SWAV: Semantics-based Workflows for Automatic Video Analysis. In *Special Session on Intelligent Workflow, Cloud Computing and Systems, (KES-AMSTA'11)*, 2011.

[23] G. Nadarajan, Y. H. Chen-Burger, and R. B. Fisher. Goal, Video Description and Capability Ontologies for Fish4Knowledge Domain. In *Special Session on Intelligent Workflow, Cloud Computing and Systems, (KES-AMSTA'12)*, 2012.

[24] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz Avila. SHOP: Simple Hierarchical Ordered Planner. In *International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 968–973, 1999.

[25] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[26] 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, 2001.

[27] W. M. P. van der Aalst and A. H. M. ter Hofstede. *Workflow Patterns Home Page*. Eindhoven University of Technology & Queensland University of Technology. http://www.tm.tue.nl/it/research/patterns. Last accessed: Apr 24th, 2012.

[28] S. van Splunter, F. M. T. Brazier, J. A. Padget, and O. F. Rana. Dynamic Service Reconfiguration and Enactment using an Open Matching Architecture. In *International Conference on Agents and Artificial Intelligence (ICAART'09)*, pages 533–539, 2009.

[29] S. A. White and D. Miers. *BPMN Modeling and Reference Guide: Understanding and Using BPMN*. Future Strategies Inc., 2008. http://www.bpmn.org. Last accessed: May 3rd, 2012.

[30] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.

 © Fish4Knowledge Consortium, 2012