Scientific workflow systems - can one size fit all?

V. Curcin, M. Ghanem
Department of Computing
Imperial College London
180 Queen's Gate, London SW7 2AZ
Email: vc100@doc.ic.ac.uk, mmg@doc.ic.ac.uk

***Abstract*—The past decade has witnessed a growing trend in designing and using workflow systems with a focus on supporting the scientific research process in bioinformatics and other areas of life sciences. The aim of these systems is mainly to simplify access, control and orchestration of remote distributed scientific data sets using remote computational resources, such as EBI web services. In this paper we present the state of the art in the field by reviewing six such systems: Discovery Net, Taverna, Triana, Kepler, Yawl and BPEL.**

We provide a high-level framework for comparing the systems based on their control flow and data flow properties with a view of both informing future research in the area by academic researchers and facilitating the selection of the most appropriate system for a specific application task by practitioners.

I. Introduction

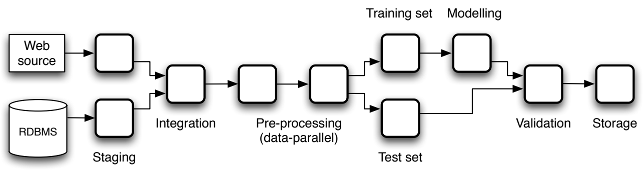

Fig. 1. Workflow example

Informally, a workflow, Figure 1, is an abstract description of steps required for executing a particular real-world process, and the flow of information between them. Each step is defined by a set of activities that need to be conducted. Within a workflow, work (e.g. data or jobs) passes through the different steps in the specified order from start to finish, and the activities at each step are executed either by people or by system functions (e.g. computer programs). Workflows are typically authored using a visual front-end or be hard-coded, and their execution is delegated to a workflow execution engine that handles the invocation of the remote applications.

Traditionally, workflow systems are split into two broad families, one for control orchestration of business processes and the other for functional style computation of data. However, the requirements of numerous applications do not fit neatly into either of those categories. This was a rationale for evolution of scientific workflow systems, that act as middleware in the scientific research process and typically have properties of both control and data workflows. Their function is to abstract over computational and data resources and enable collaboration between researchers, a task which requires both aspects. The question we are interested in is whether any single workflow system (scientific or non-scientific) can be relied on to cover the scope of requirements from different domains.

This paper approaches the problem by analysing leading scientific and non-scientific workflow systems, exposing their handling of control and data constructs, with the view of informing future research and also facilitating the selection of the most appropriate system for a specific application task.

As a start, Discovery Net [1] system will be presented to illustrate the architectural and implementation complexity associated with a full workflow system. Then, three other main scientific workflow systems, Taverna [2], Triana [3] and Kepler [4], will be described, followed by two workflow languages aiming to be a generic solution across both business and scientific domains. First of those, YAWL [5] is a theoretical workflow system based on the Petri Net paradigm that has been designed to satisfy the full set of workflow patterns, under the assumption that this will satisfy the needs of both communities. Second, BPEL [6] is the accepted standard for business process orchestration, with several attempts being made to adapt it for use in scientific settings, most notably by the OMII initiative [7].

II. Discovery Net

The Discovery Net system has been designed around a scientific workflow model for integrating distributed data sources and analytical tools within a grid computing framework. The system was originally developed as part of the UK-e-Science funded project Discovery Net (2001-2005) [8] with the aim of producing a high-level application-oriented platform, focused on enabling the end-user scientists in deriving new knowledge from devices, sensors, databases, analysis components and computational resources that reside across the Internet or grid.

Its dedicated set of components for data mining has been used as a basis for numerous cross-domain projects. These include Life Sciences applications [9], [10], Environmental Monitoring [11] and Geo-hazard Modelling [12]. Many of the research ideas developed within the system have also been incorporated within the InforSense KDE system [13], a commercial workflow management and data mining system that has been widely used for business oriented applications. A number of extensions have been based on the research outputs of the EU-funded SIMDAT [14] project.

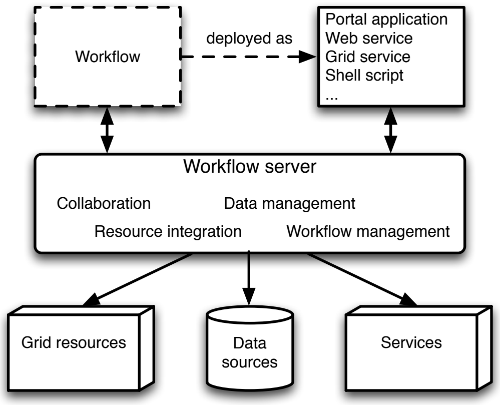

Fig. 2. Discovery Net concept

A. System overview

Figure 2 provides a high-level overview of the Discovery Net system. The system is based on a multi-tier architecture, with a workflow server providing a number of supporting functions needed for workflow authoring and execution, such as integration and access to remote computational and data resources, collaboration tools, visualisers and publishing mechanisms.

The design of the system targets the domain experts, i.e. scientific and business end users, rather than distributed and grid computing developers. The aim is that users can develop and execute their distributed data mining workflows through a drag-and-drop authoring client. The workflows created in this way can also be executed from specialized web-based interfaces. The three tier architecture model on which the different versions of Discovery Net system were based on, is presented in Figure 2. The implementation of the system itself has evolved over the past few years from a prototype targetted to specific projects to an industrial strength system widely used by commercial and academic organizations.

B. Workflow representation in DPML

Within Discovery Net, workflows are represented and stored using DPML (Discovery Process Markup Language) [15], an XML-based representation language for workflow graphs supporting both a data flow model of computation (for analytical workflows) and a control flow model (for orchestrating multiple disjoint workflows).

Each node in a DPML workflow graph represents an executable component (e.g. a computational tool or a wrapper that can extract data from a particular data source). Each component has a number of parameters that can be set by the user and also a number of input and output ports for receiving and transmitting data, as shown on Figure 3. Each arc in the graph represents a connection from an output port, namely the *tail* of the arc, to an input port, namely the *head* of the arc. A port is connected if there is one or more arcs from/to that port. Metadata of the node describes the input and output ports, including the type of data that can be passed to the component and parameters of the service that a user might want to change. Such information is used for the verification of workflows and to ensure meaningful chaining of components. A connection between an input and an output port is only valid if the types are compatible, which is strictly enforced.

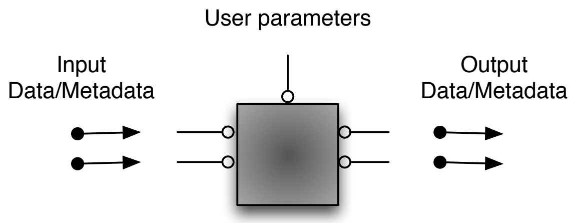

Fig. 3. Structure of a component in Discovery Net

C. Service deployment

One of the key contributions of Discovery Net to e-Science community is its automated mapping of workflows into reusable services that present a user–friendly web interface for reserachers to execute analytical workflows and inspect their results [15]. Deployment, as this process is known, consists of specifying a subset of node inputs, node parameters and an output to form a single task, which user can execute with a single click, or a single service invocation. This method was successfully used in multiple academic and industrial settings, and has inspired a number of other workflow and analytical tools.

D. Control flow

Discovery Net is primarily a data flow system, with a control model added as a top level co-ordination layer. As opposed to data flow graphs, nodes within the control graphs represent special constructs that orchestrate execution, control iteration and branching logic. These graphs can be cyclic in their definition and communication between them is based on passing control tokens rather than data. The execution of Discovery Net control flow graphs is based on a *push* paradigm where workflow components are invoked left-to-right and nodes with multiple output ports can decide on which port(s) they will place the tokens, hence determining the direction of further execution. Control flows in Discovery Net are used mainly to coordinate disconnected workflows whose ordering is driven by business logic, rather than an explicit data dependency. Therefore, control flow nodes may contain within them data flows (deployed as services) that are invoked every time the node executes. No communication exists between the two layers, nor between two data flow nodes in a control flow.

E. Data flow

Discovery Net supports a data *pull* model for data flow graphs with a workflow acting as an acyclic dependency graph. Within this model a user requests execution of one the end-point nodes in the graph, which then initiates the execution of all the required preceeding nodes.

Discovery Net is a typed workflow system, which not only ensures that workflows can be easily verified during their construction, but also helps in optimising the data management by organizing components by their inputs or outputs. The default type for the data in the system is a relational table, consisting of rows of tuples of column values. However, for the purpose of supporting additional applications, several specific data models were added: a bioinformatics data model for representing gene sequences [10], an image collection [11] and a mark-up model for text mining based on the Tipster architecture [16]. Each model has with it an associated set of data import and export components, as well as dedicated visualisers which integrate with the generic import/export/visualisation tools already present in the system. As an example, chemical compounds represented in the widely used SMILES format can be imported inside data tables where they can be rendered adequately either using a 3-d representation or its structural formula.

III. Taverna

Taverna is a common name used for a scientific workflow system comprising Taverna Workbench graphical workflow authoring client, together with SCUFL [17] workflow representation language, and Freefluo [18] enactment engine. It is a key part of the ^{my}Grid e-Science initiative, that includes additional components such as a service directory, ontology–driven search tools, data and metadata repositories and others.

The primary aim of Taverna is to satisfy the needs of bioinformaticians who need to build scientific workflows from numerous remote web services. Therefore, a significant effort in Taverna went towards harvesting and organizing these web services into a usable collection of components.

Due to this reliance on components coming from different autonomous service providers, Taverna has been designed to operate in an open-world setting, where no common data format is assumed beyond XML. Consequently, the user is expected to resolve these formats manually when composing services, a process known as shimming. In addition to these, a set of generic component types is provided for fast integration or development of new components based on WSDL files, Java code, Soaplab [19] services, Seqhound [20] REST services and existing SCUFL workflows (form of embedding).

SCUFL (Simple Conceptual Unified Flow Language) is a language for representing workflows as Directed Acyclic Graphs. A workflow may have zero or more formal input parameters. These are represented in Scufl as sources. Sources must have a unique name within the namespace of the other sources in a Scufl document. Similarly, the formal outputs of a Scufl workflow are represented as sinks. Sinks can be associated with metadata in the form of any number of mime types to aid with visualisation and must have a unique name within the namespace of sinks within a Scufl document.

The basic execution units in SCUFL are processors which may be regarded as a function of some set of input data to a set of output data, represented as ports on the processor. Two types of link are present, data links through which data flows between node ports, and coordination links that act as additional ordering separate from data dependency.

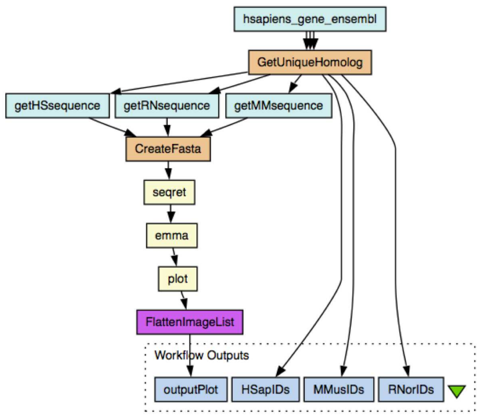

Fig. 4. Taverna workflow

The execution of a SCUFL workflow is performed in a push manner, starting from the sources and finishing when all sinks have either produced their outputs or failed. The user can configure whether the workflow can execute partially, allowing some sinks to complete even if another one failed. The workflow subset that is being executed is determined at runtime by calculating the path from the source nodes, and any other nodes with no dependencies, to the sink nodes.

A. *Control flow*

SCUFL is an example of a dataflow language, with some additional constructs typically associated with control flows. The idea is that processors in SCUFL may have an effect on the execution environment which is separate from inputs and outputs, which introduces the requirement for explicit ordering constraints not based on the data dependency between processors.

For example, consider a workflow, one of whose tasks is to update a database and perform indexing on it, before proceeding with the other tasks. The indexing task is not connected to subsequent tasks by a data dependency, however it is a requirement that it needs to run. In order to achieve this, a coordination link is created based on the gate constraint that must be satisfied before a processor can effect a particular state change – in this example completion of the indexing task. Concurrency constraints are frequently used in Taverna for dealing with stateful Grid services.

SCUFL also has a conditional construct, based on the notion of having data passed to multiple components, all of which are guaranteed (by the workflow author) to fail apart one. This corresponds to the *case* structure in classic programming languages.

The coordination links and the conditional construct are the only control structures available in Taverna. A loop construct is not present, however a limited form of iteration is available by wrapping up date objects into lists and specifying the iteration strategy.

B. Data flow

At the basic level, two processors connected with a data link represent function composition. So, if a processor A is connected to B, the result of B will be $f_B(f_A)$. Complexity arises from Taverna's generality.

One of the guiding principles of SCUFL is that it is data–agnostic [21]. However, this still leaves the problem of how to distinguish operations which work on one input object, from the ones requiring a collection of those objects. The first step of the solution was to introduce lists and trees, as data structures in SCUFL. Since the processors are mainly remote web services, a mechanism called *configurable iteration* was developed to apply the same component to the input, or a collection of inputs.

Configurable iteration amounts to specifying the strategy for function application. For example, if a processor with function f takes in one input a, the default output is $f(a)$. However, if the workflow designer knows that a is in fact a list $[a_1, a_2, ..., a_n]$, he can specify that function be applied as a map, and produce the output $[f(a_1), f(a_2), ..., f(a_n)]$.

In the cases where the input to the function are two lists $[a_1, a_2, ..., a_n]$ and $[b_1, b_2, ..., b_n]$, the function can either be applied as a *dot-product*, $[f(a_1, b_1), f(a_2, b_2), ..., f(a_n, b_n)]$, or as a *cross-product*, $[f(a_1, b_1), f(a_1, b_2), ...f(a_1, b_n), f(a_2, b_1), f(a_2, b_2), ..., f(a_2, b_n), ..., f(a_n, b_n)]$.

The goal of this approach is to minimize the number of processors available in the system and make them as applicable as possible to various data structures. This reflects the open-world assumption at the basis of Taverna. This differs from closed-world systems such as Discovery Net, which, in addition to generic capabilities, has privileged relational operations which are used to explicitly define dual input operations that are considered useful.

IV. TRIANA

Triana is a visual workflow-based problem solving environment, developed at Cardiff University. Originally, it was associated with a gravitational wave detection project, GEO 600 [22] and used as a rapid analysis tool for wave data. Subsequently, it has been extended to incorporate a range of modules, such as peer-to-peer component communication, GAT [23] integration for Grid services, GAP [24] integration with Web services and JXTA service invocation, all with the purpose of providing better integration with existing Grid technologies. Despite the inclusion of a multitude of remote components, it retains the core set of local processing units as well to allow for local data transformations and visualisations.

The functional component in Triana is called a *unit*. Units are connected by directed *cables* coming in and out of their ports to form workflows, similarly to other scientific workflow systems presented here. A higher-level group unit exists for the purpose of embedding workflows into each other as units.

Unlike Taverna, Triana supports several data models, reflecting its use in multiple application domains. Over 500

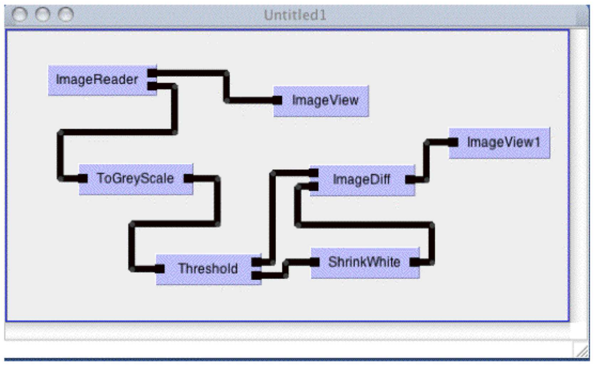

Fig. 5. Triana workflow

components [25] were developed for signal, image and audio processing and statistical analysis, grouped in toolboxes of related components. In addition to these, a generic set of components was developed for integration of Java code, legacy appications, WS–RF, P2P or WSDL web services. The cable implementations are resolved at runtime based on the types of units that are connected. For example, a cable between two local units will cause a file to be moved from one location in the filesystem to another, while the cable between two remote tools will initiate a GridFTP transfer.

One notable capability of Triana is to modify and republish any node. The source code for each node in the toolbox can be viewed, modified and recompiled within the environment, allowing for rapid development of new components in pure Java. Similar techniques in other workflow systems utilize scripting languages, Groovy [26] in Discovery Net and NetBeans [27] in Taverna.

A. Control flow

Execution in Triana is push–based, with each component's output sent down the cables to receiving components which then start executing. Despite being a data–flow system, Triana provides support for control flow through special messages that trigger control between units. In addition to these, there are dedicated nodes for branching, which pass data only to one of the recipients, and looping. These constructs can be freely combined with functional components. The conditional selection is based on the coupled use of *If* and *Switch* components. *If* passes the data to one of two nodes based on some condition, while *Switch* selects one of two inputs also based on some condition.

B. Data flow

The default data flow in Triana is from the source unit to the destination unit, representing function composition. However, addition of numerous control structures breaks the functional metaphor through non–determinism. Effectively, a history of executed components represents the functional composition that was performed.

The standard data flow mechanism completes the execution of one node before passing the result to the next one. This is not the only supported paradigm, and streaming is possible using a set of dedicated components such as *Sequence*, *Block*, *Merge*, and others. Through these constructs it is possible to schedule parallel execution of several units in the graph on different subsets of data.

V. Kepler

Kepler is a scientific workflow construction, composition, and orchestration engine, evolved from Ptolemy II [28], an actor-oriented modeling tool meant primarily for embedded and real-time system design. Kepler's focus is on data analysis and modelling, which influenced the design in that it is suitable for modelling processes in a wide variety of scientific domains, from physics via ecosystems to bioinformatics web services.

Instead of trying to provide a generic semantic for all possible types of processes encountered in these domains, Kepler separates the execution engine from the workflow model, and assigns one model of computation, *director*, to each workflow.

The workflow components in Kepler are represented by *actors* which represent operations or data sources, with a number of ports, whether input, output, or mixed, that act as end-points for connections that transport tokens. The simplest interaction consists of an actor consuming one data token on each input port and producing one token on each output port whenever it executes ("fires"). However, there are numerous cases where more than one token may be consumed (or produced) for each execution. An example is the "Sequence to Array" actor, which consumes a certain number of input tokens specified in advance and then outputs a single array token.

Directors are the key concept in Kepler. While actors and relations together constitute a workflow model, the directors form the execution model, or Model of Computation (MoC). In this setup, intelligence of an actor stretches as far as knowing its inputs, the operation to be performed on them and what outputs to produce. The decision when to schedule the execution of each actor is left to the director. For example, an addition operation can accept input data delivered by any of a number of mechanisms, including discrete events, rendezvous, and asynchronous message passing. Therefore, depending on the director used, the actors may have separate threads of control, or they may have their executions triggered by the availability of new input, in a more conventional dataflow manner. This architecture, in which components are agnostic to the manner in which they are executed, is referred to as *behavioural polymorphism* [29].

Furthermore, in order to increase reusability, Kepler actors are *data polymorphic* in the sense that they can be applied to multiple data types on inputs. That way, the same component can be used to perform, for example, addition of integers, real numbers and string concatenation.

A. Control flow

The execution of actors by the directors is centered around the notion of tokens. When an actor receives a token, it runs the required number of times, and as it does it fires new tokens with the resulting data on the output port. The execution consists of preinitialising all components, type-checking compositions, running each node and finalizing the task. When a node runs, it initializes any resources it requires and then continues firing whenever it receives a token.

Kepler's architecture allows new execution semantics to be plugged in by the users, defined using programmatic manipulation of objects in the system. Below are described the four core directors.

SDF (Synchronous Data Flow) is characterized by fixed token production and consumption rates per firing. The actor is invoked as soon as all inputs have data, which is possible to know since all actors have to declare their token production before the execution. Therefore, the order of execution is statically determined from the model, and components cannot change the routing of tokens during execution. This type of semantics is suitable for modelling linear dataflows such as mathematical calculations, or tabular manipulation.

PN (Process Network) is a derestricted variant of SDF, in that the actor is invoked when the data arrives. However, there is no requirement that *all* data has to be present, which results in a more dynamic environment, where actors are executing in parallel and sending each other data when and if needed. The tokens are created on output ports whenever input tokens for an actor are available and the outputs can be calculated. The output tokens are then passed to connected actors where they are held in a buffer until that next actor can fire. The workflow is thus driven by data availability. This modelling is naturally suited to messaging environments, such as communicating web and Grid services, and parallel processing on distributed systems.

The CT (Continuous Time) director introduces the notion of time that is affixed to tokens in order to perform system simulations. The system is usually described in terms of differential equations, and the start conditions, which are then used to predict the state at some specified time in the future. For example, given some equation, the system can calculate the value of the function at each step, display it in some dynamic visualiser, and pass it back into the function so it can calculate the next step. The data tokens that are passing through the system then have a timestamp that the director is using to determine the step and the stop condition.

Similarly to CT, the DE (Discrete Event) director is working with timestamps; however they are not used to approximate functions and schedule executions, but to measure average wait times and occurrence rates. Therefore, the actors determine when tokens are sent, as in PN, but they also capture the time dimension of the token passing process. Typical use of this sort of workflow is to model the bus/passenger problem [30] and similar modelling questions.

Kepler supports hierarchical embedding of a workflow into another workflow, depending on the compatibility of their directors. The allowed combinations are determined by two factors: the requirements of the director from underlying workflows under its control, and the semantics exported by the director to the actor within which it is placed. Based on the semantics, each director can be classified as *strict*, *loose* or *loosest*, and the embeddings are allowed only when the inner director is at least as strict as the outer one. Detailed analysis can be found in [31].

B. Data flow

The tokens in Kepler contain data. Whenever a node execution is initiated, the node reads its inputs from the incoming token and places the results into the outgoing one. The director that is assigned to the given workflow handles the order in which the calculations are executed, so they are not a direct consequence of the functional dependencies between components. Therefore, despite having data passed around the workflow, Kepler does not have a purely functional composition model, reflecting the dual nature of Kepler's aim, to support both analysis processes and process simulations. The lack of functional composition also makes it less applicable to analytical tasks with a strong prototyping slant, where decisions as to which component to apply next are performed dynamically [4].

VI. YAWL

YAWL, or Yet Another Workflow Language, was not designed as a scientific workflow system and unlike other systems described so far, it has its origins in theoretical work and was not developed for the purposes of any single application project. What makes it interesting for the purpose of this review is that it was designed with the purpose of being a generic workflow tool, equally applicable to scientific work and organization of business processes, reflecting the philosophy that there is no significant difference between the two [32].

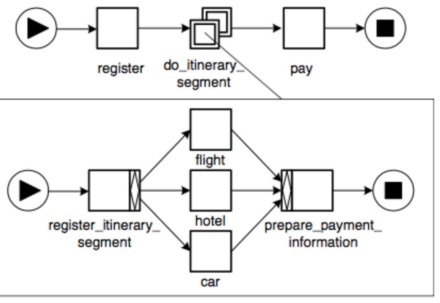

Fig. 6. YAWL workflow

The requirements for YAWL came from the work on workflow patterns [33] which delivered a comparison of a number of (mostly business) workflow systems and attempted to formalize each feature as a pattern. The goal of YAWL was to provide a language that would capture the superset of all these capabilities. Realizing that all the patterns described can be implemented in high-level Petri nets, they were used as a formal basis for the language. While the control flow perspective of YAWL is the most used one, the language also supports the data and resource perspectives.

Workflow in YAWL, as shown in Figure 6, is a set of tasks, conditions and flows between them. Unlike Petri Nets, tasks can be connected to each other directly, without a condition in–between, which is interpreted as an implicit condition that automatically succeeds. A task in a workflow can be either atomic or composite, with composite tasks containing another workflow, or *extended workflow net* in YAWL terminology, forming a tree like structure of hierarchical workflows.

Each workflow, whether top-level or inner, must have a single input and a single output condition, thereby simplifying verification and soundness computation. Each task can have multiple instances specified, and be restricted with upper and lower bounds, instance threshold (being able to trigger some condition based on the number of instances of a task) and a parameter that specifies whether the number of instances is static during task execution or dynamic, ie. can change depending on the execution result.

A. Control flow

In YAWL, the basic Petri Net model was extended with features to support three main tasks: multiple process instances, advanced synchronization of tasks (where several tasks can trigger the OR-join) and removal of tokens from the net, used for cancellation.

There are six explicit branching constructs: a split and a join of AND, XOR, and OR, which model every legal data routing through the workflow. Due to the nature of splits the execution path through the workflow is determined dynamically at runtime, as opposed to being apriori statically determined.

Conditional branching is achieved through XOR and OR splits and joins, that have to be paired up, forming a conditional block. Looping is performed through the use of OR join and a state node (in the style of Petri Nets) that can decide what to fire.

B. Data flow

All data in YAWL is represented as XML documents with XPath and XQuery used for any transformation operations. The transfer of data between components is achieved via a shared workflow state containing variables. There is no concept of data being passed down the connections between nodes. Two types of data transfers exist, internal and external, both using XQuery.

Internal transfers are always performed between the tasks and their workflows, since all variables inside the tasks are internal to that task and cannot be shared with another task. Data sharing between tasks is performed via task parameters, which are associated with an environment variable in the workflow. So, in order to communicate some data between tasks A and B, task A has to register its variable as the output parameter, and pass it to some global workflow variable N, which task B will take as its input parameter.

External transfers operate, at run–time, between global variable and the user or the global variable and an external component, such as a web or Grid service. The exact process of retrieving or sending the data is handled by the YAWL enactment engine.

C. *YAWL as a scientific workflow system*

The idea at the heart of YAWL is that the difference between scientific and business workflows is immaterial [32]. While it is certainly true that two areas can inform each other and a number of research topics are common to both, the authors' view is that the key difference between the two lies in the scientific workflows' focus on *innovation*, rather than *automation*. This is not a cosmetic feature, but has a more fundamental impact on the design of workflow systems that can adequately support high levels of abstraction and transparently integrate a wide variety of resources and computational models.

Considering the data aspect, which is key to any scientific workflow system, the role of data flow in YAWL is not to perform computations, but to perform data transformations as a possible side-effect of workflow execution. So, despite its very solid theoretical framework, and a wide spectrum of capabilities, it provides no workflow–style view of computation, and is therefore not applicable as a scientific workflow system.

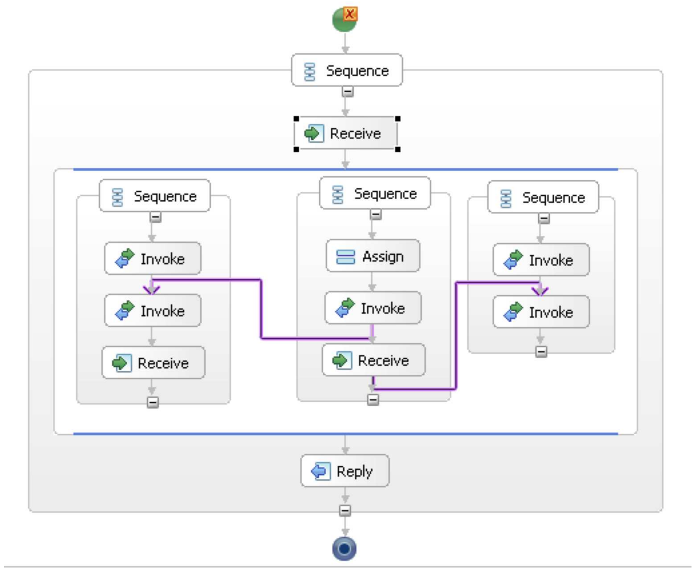

Fig. 7. BPEL workflow

VII. BPEL

BPEL4WS (Business Process Execution Language for Web Services) was established in 2002, when it superseded and amalgamated IBM's WSFL (Web Services Flow Language) and Microsoft's XLANG. An important aspect of BPEL is its transactionality, reflected in advanced error handling mechanisms. While it is not a graphical language, many of BPEL constructs can be represented graphically. BPEL 2.0 came out in 2004 and is the most widely accepted version of the language.

At the core of BPEL design are web services. Each BPEL workflow can be observed as a web service in its own right. Hierarchical composition is supported through invocation of other BPEL workflows as services. The workflow consists of *partner links*, which are external entities (other web services or users) that are going to communicate with the workflow, *variables* which store data used in the workflow, and a *control structure*.

An example of a BPEL workflow is shown in Figure 7. Workflows typically start with a $\{receive\}$ activity and ends with a $\{reply\}$ indicating that the workflow was invoked as a service, and returns some value to its invoker. The $\{invoke\}$ operation allows calls to a remote service, either in a synchronous or an asynchronous way.

A. *Control flow*

BPEL offers two ways of specifying relationships between activities. One is by using the control constructs: *If*, *Pick*, *ForEach*, *RepeatUntil* and *While*, which implement different forms of conditional branching and looping. The other is by using activity containers such as *sequence* and *flow*, which schedule several activities either sequentially or in parallel. Iterated activities cannot be parallelized.

In addition to this, tasks in BPEL can be composed using graph–like links. In those cases, an activity with multiple links coming in will wait for all the messages to arrive before proceeding, and once it completes, it will send out messages on all its outgoing links.

B. *Data flow*

All data in BPEL is represented by XML documents, and operated on using XPath and XQuery. No data operations are present in BPEL, they are left to the underlying implementation of individual services, but an $\{assign\}$ task is present to link the service inputs and outputs to variables in the workflow, similarly to the way variable space is used in YAWL.

C. *BPEL as a scientific workflow system*

In [34], it was argued that scientists prefer to think of their analysis on a higher level than that of web service ports, and therefore BPEL is not at the right level of abstraction for scientific purposes. Still, several attempts have been made at adapting BPEL for scientific workflows. OMII-BPEL [35], led by the Open Middleware Infrastructure Institute [7], produced an integration of Active BPEL and BPEL Designer tools, enhancing them with some Grid-specific components. GPEL4SW [36] (Grid Process Execution Language for Scientific Workflows) has been developed at the University of Indiana with the aim of specializing BPEL to running Grid processes.

Sedna [37] is the effort to simplify the process of construction of workflows with BPEL, and make it more accessible to domain users with little knowledge of technical detail. This is done through a graphical construction environment for BPEL workflows, and several extensions to the language, namely: indexed flows (which allow parallelization of iteration), different modes of reducing components (through hierarchical abstraction or reuse of existing components in other workflows) and macros, which automate message interactions.

VIII. Feature comparison of reviewed systems

Following the presentation of these six systems, let us now see how they compare with respect to some syntactical features, and control and data flow aspects.

A. Syntactical features

Several syntactical features are relevant to the discussion. The ability of a node to have multiple ports that produce different output types, is present in all of the systems apart from Triana. Shared variable space is not present in the data flow systems listed here, since the communication of data happens through component links, however it is used in BPEL and YAWL to exchange values in absence of data flows. Finally, Discovery Net, Triana and Kepler attach type information to the data being passed and ensure that component composition is type-safe.

B. Control behaviour

Elements of control and data separation exist in these workflow systems in three distinctive flavours: as different types of workflow, as different subsets of nodes that implement both control and data functionality in the same workflows, or as different types of links that transports control or data.

- **Separate layers.** Discovery Net implements control flows as separate entities from the data flows, with no interaction involved. BPEL and YAWL only provide control-style workflows. Kepler defines control using directors that determine the workflow node behaviour.
- **Separate nodes.** Triana has a set of control nodes which are used to achieve branching, parallelism and looping.
- **Dedicated control and data links.** Taverna uses control links, which pass no data, to synchronize execution of components with no data dependency. No looping is present.

C. Data behaviour

Data flow execution can be orchestrated in a data-driven or model-driven manner, also known as push and pull semantics, or it can be choreographed by giving each node detailed instructions on how to behave when data becomes available.

- **Data-driven orchestration.** Taverna and Triana execution proceeds by running all the nodes with no predecessors and continues until there are no more nodes left to run.
- **Model-driven orchestration.** Discovery Net performs only the operations needed to produce the required result, executing the subset of the graph that is relevant to the output.
- **Choreographed behaviour.** Each node is given instructions on how to behave with regards to data. In Kepler, this behaviour is common to all the nodes in the layer and defined by the director. In YAWL, it is specified by the node interaction properties. In BPEL, it is left to the individual service.

D. Embedding

When it comes to combining control and data aspects, Discovery Net, Triana and Kepler all provide such mechanisms through embedding constructs in which a workflow of one type is embedded into the workflow of another type.

- **Data as implementation of control execution.** Discovery Net allows the control flow to perform the orchestration of individual data flows by embedding them inside the control nodes. The logic is then dictated by the control flow, with any data transformations performed in the internal data flows.
- **Control flow as computational pattern.** Triana associates explicit coordination logic with an embedded workflow by using scripts. Primary use of this mechanism is to create advanced looping constructs, but more generally it enables creation of any imperative logic. The coordination is performed by script code, not a workflow, manipulating input and output values between the embedded and the outside workflow.
- **Multiple process semantics.** Kepler aims to generalize all notions of computations applicable to graph representations by defining a fixed number of workflow semantics, embodied in directors, each with a particular process behaviour [38]. For example, an enclosing director may impose the restriction that the internal component has to return a concrete value or a clear termination signal, while some others may just require it to start, without being concerned whether it ever completes.
- **Embedding of the same types.** Embedding of a workflow into another workflow of the same type is present in Discovery Net, Taverna, BPEL and YAWL as a grouping operation, which hides away some of the complexity of the graph. In addition to being just a visual feature, Discovery Net and BPEL use the grouped entity for publishing the functionality as a service.

IX. Conclusions and future work

In this paper, we reviewed four of the most popular scientific systems, one purely theoretical system that has not evolved from the needs of any particular research project, and a generally accepted standard system for business process workflows. The differences between them were in their approach to handling data, with scientific workflow systems treating their data structures as parts of the workflow representation, rather than as disconnected objects. The role of embedding in combining the two semantics was also presented.

The variety of control and data elements that are available to workflow authors and users indicate that scientific workflow systems will continue being developed for various domains and guided by those domains' needs. Due to this, it is highly unlikely that standardization will occur on any one system, as it did with BPEL in the business process domain. Therefore, modelling and analysing the process and data capabilities of workflow systems in a framework independent of any particular implementation is a valid and necessary research goal.

References

[1] M. Ghanem, V. Curcin, P. Wendel, and Y. Guo, "Building and using analytical workflows in discovery net," in *Data mining on the Grid*, W. Dubitzky, Ed. John Wiley and Sons, 2008.

[2] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn, "Taverna: A tool for building and running workflows of services," *Nucleic Acids Research*, vol. 34, pp. W729–W732, 2006, web Server Issue.

[3] I. Taylor, M. Shields, I. Wang, and A. Harrison, "Visual Grid Workflow in Triana," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 153–169, September 2005. [Online]. Available: http://www.springerlink.com/openurl.asp?genre=article&issn=1570-7873&volume=3&issue=3&spage=153

[4] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, 2006.

[5] W. van der Aalst and A. Hofstede, "Yawl: Yet another workflow language," 2002. [Online]. Available: citeseer.ist.psu.edu/vanderaalst03yawl.html

[6] 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," http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, BEA Systems, IBM, Microsoft, SAP, Siebel Systems., Tech. Rep., 2003.

[7] J. Bradley, C. Brown, B. Carpenter, V. Chang, J. Crisp, S. Crouch, D. D. Roure, S. Newhouse, G. Li, J. Papay, C. Walker, and A. Wookey, "The omii software distribution," in *All Hands Meeting 2006*, 2006, pp. 748–753. [Online]. Available: http://eprints.ecs.soton.ac.uk/13407/

[8] A. Rowe, D. Kalaitzopoulos, M. Osmond, M. Ghanem, and Y. Guo, "The discovery net system for high throughput bioinformatics," *Bioinformatics*, vol. 19, no. 90001, pp. 225i–231, 2003.

[9] M. Ghanem, Y. Guo, H. Lodhi, and Y. Zhang, "Automatic scientific text classification using local patterns: KDD CUP 2002 (task 1)," *SIGKDD Explor. Newsl.*, vol. 4, no. 2, pp. 95–96, December 2002.

[10] V. Curcin, M. Ghanem, Y. Guo, A. Rowe, W. He, H. Pei, L. Qiang, and Y. Li, "It service infrastructure for integrative systems biology," in *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 123–131.

[11] M. Richards, M. Ghanem, M. Osmond, Y. Guo, and J. Hassard, "Grid-based analysis of air pollution data," *Ecological Modelling*, vol. 194, pp. 274–286, 2006.

[12] Y. Guo, J. G. Liu, M. Ghanem, K. Mish, V. Curcin, C. Haselwimmer, D. Sotiriou, K. K. Muraleetharan, and L. Taylor, "Bridging the macro and micro: A computing intensive earthquake study using discovery net," in *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 68.

[13] (2008) Inforsense ltd. [Online]. Available: http://www.inforsense.com/

[14] C. Upstill and M. Boniface, "Simdat," *CTWatch Quarterly*, vol. 1, no. 4, pp. 16–20, November 2005. [Online]. Available: http://eprints.ecs.soton.ac.uk/11622/

[15] J. Syed, M. Ghanem, and Y. Guo, "Discovery processes: Representation and reuse," in *Proceedings of First UK e-Science All-hands Conference, Sheffield, UK*, 2002.

[16] M. Ghanem, A. Chortaras, and Y. Guo, "Web Service programming for biomedical text mining," in *SIGIR Workshop on Search and Discovery in Bioinformatics held in conjunction with the 27th Annual International ACM SIGIR Conference, Sheffield, UK*, July 2004. [Online]. Available: http://pubs.doc.ic.ac.uk/web-service-text-mining/

[17] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, C. Goble, A. Wipat, P. Li, and T. Carver, "Delivering web service coordination capability to users," in *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. New York, NY, USA: ACM, 2004, pp. 438–439.

[18] (2008) Freefluo enactment engine. [Online]. Available: http://freefluo.sourceforge.net

[19] S. Pillai, V. Silventoinen, K. Kallio, M. Senger, S. Sobhany, J. G. Tate, S. S. Velankar, A. Golovin, K. Henrick, P. Rice, P. Stoehr, and R. Lopez, "Soap-based services provided by the european bioinformatics institute." *Nucleic Acids Research*, vol. 33, no. Web-Server-Issue, pp. 25–28, 2005.

[20] K. Michalickova, G. D. Bader, M. Dumontier, H. Lieu, D. Betel, R. Isserlin, and C. W. V. Hogue, "Seqhound: biological sequence and structure database as a platform for bioinformatics research." *BMC Bioinformatics*, vol. 3, p. 32, 2002.

[21] I. J. Taylor, E. Deelman, and D. B. Gannon, *Workflows for e-Science: Scientific Workflows for Grids*. Springer, December 2006.

[22] R. Balasubramanian, S. Babak, D. Churches, and T. Cokelaer, "Geo600 online detector characterization system," *Classical and Quantum Gravity*, vol. 22, p. 4973, 2005. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:gr-qc/0504140

[23] G. Allen, K. Davis, T. Dramlitsch, T. Goodale, I. Kelley, G. Lanfermann, J. Novotny, T. Radke, K. Rasul, M. Russell, E. Seidel, and O. Wehrens, "The gridlab grid application toolkit," in *HPDC '02: Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02)*. Washington, DC, USA: IEEE Computer Society, 2002, p. 411.

[24] I. Taylor, M. Shields, I. Wang, and O. Rana, "Triana Applications within Grid Computing and Peer to Peer Environments," *Journal of Grid Computing*, vol. 1, no. 2, pp. 199–217, 2003. [Online]. Available: http://journals.kluweronline.com/article.asp?PIPS=5269002

[25] I. Taylor, M. Shields, I. Wang, and A. Harrison, "The Triana Workflow Environment: Architecture and Applications," in *Workflows for e-Science*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds. Secaucus, NJ, USA: Springer, New York, 2007, pp. 320–339.

[26] R. Winder, "Hello groovy!" *CVU*, vol. 18, no. 3, pp. 3–7, 2006.

[27] T. Boudreau, J. Glick, and V. Spurlin, *NetBeans: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2002.

[28] C. Brooks, E. Lee, X. Liu, S. Neuendorffer, and H. Z. e. Y. Zhao, "Heterogeneous concurrent modeling and design in java (volume 2: Ptolemy ii software architecture)," EECS Dept., UC Berkeley, Tech. Rep. 22, July 2005. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/63.html

[29] E. A. Lee and Y. Xiong, "A behavioral type system and its application in ptolemy ii," *Form. Asp. Comput.*, vol. 16, no. 3, pp. 210–237, 2004.

[30] T. Nagatani, "Interaction between buses and passengers on a bus route," *Physica A Statistical Mechanics and its Applications*, vol. 296, pp. 320–330, Jul. 2001.

[31] A. Goderis, C. Brooks, I. Altintas, E. A. Lee, and C. Goble, "Heterogeneous composition of models of computation," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2007-139, November 2007.

[32] Y. Chen and van Aalst, "On scientific workflows," *IEEE Computer Society's Technical Committee for Scalable Computing*, 2007. [Online]. Available: citeseer.ist.psu.edu/plotkin03origins.html

[33] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, pp. 5–51, 2003.

[34] M. Addis, J. Ferris, M. Greenwood, P. Li, D. Marvin, T. Oinn, and A. Wipat, "Experiences with e-science workflow specification and enactment in bioinformatics," in *e-Science All Hands Meeting 2003*, S. Cox, Ed., 2003, pp. 459–466.

[35] B. Wassermann, W. Emmerich, B. Butchart, N. Cameron, L. Chen, and J. Patel, "Sedna: A BPEL-based environment for visual scientific workflow modelling," in *Workflows for eScience - Scientific Workflows for Grids*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds. Springer Verlag, 2006.

[36] A. Slominsky, "Adapting BPEL to Scientific Workflows," in *Workflows for e-Science*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds. Springer, New York, 2007, pp. 208–226.

[37] B. Wassermann, W. Emmerich, B. Butchart, N. Cameron, L. Chen, and J. Patel, "Sedna: A BPEL-based environment for visual scientific workflow modelling," in *Workflows for eScience - Scientific Workflows for Grids*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds. Springer Verlag, 2006.

[38] A. Goderis, C. Brooks, I. Altintas, E. A. Lee, and C. A. Goble, "Composing different models of computation in kepler and ptolemy ii." in *International Conference on Computational Science (3)*, ser. Lecture Notes in Computer Science, Y. Shi, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, Eds., vol. 4489. Springer, 2007, pp. 182–190.