On the equivalence of specific control flow and data flow patterns with use cases

Akos Balasko
Institute for Computer Science and Control
Hungarian Academy of Sciences
Budapest, Kende str. 13-17
Email: balasko@sztaki.hu

Peter Kacsuk
Institute for Computer Science and Control
Hungarian Academy of Sciences
Budapest, Kende str. 13-17
Email: kacsuk@sztaki.hu

***Abstract*—Although many of workflow languages use workflow patterns in various aspects such as control or data, the implementation of these patterns are quite different, which makes the workflow language interoperability really difficult. This paper introduces compositions of specific data and control patterns and then proves their equivalence, which benefits that control structures can be replaced by data patterns and vice versa, independently from the implementation itself. To confirm the necessity of our results we introduce use cases based on community workflows that apply data patterns to trigger control structures. As WS-PGRADE/gUSE workflow management system became a commonly used general framework for workflow development by various scientific communities, the use cases are created in there, to facilitate using our solutions in other workflows.**

I. Introduction

At present workflow management systems are key tools for scientific communities to be able to do research by defining problems and algorithms on a higher level. Using workflows scientists can connect applications to each other to make them interact automatically. Therefore workflow languages together their syntax and semantics are quite relevant to have a clear view, what can be defined in a particular language and what possibilities are not supported. Unfortunately so many workflow languages are at service so many difference are within their expression power. For instance a structured loop is supported by BPEL[10] or by Taverna[6], but not by EPC[8]. Van der Aalst et. al. investigated many existing workflow languages and workflow management systems identifying many general patterns of workflow structures in various perspectives, such as control flows [4] or data-flow concept [5]. Their results are quite important, in many cases they specify a pattern in a generic way, but in most cases the investigated workflow languages' implementation just support a construct to achieve the behaviour of a pattern, not the pattern itself directly. Of course this is not really confusing till a community uses only one workflow language, the scientist can learn the ”dialect” of the patterns implemented. But when they would like to use more languages, for instance to be able to create structures that are not supported by the old one, these implementation differences make the language interoperability really complicated, if not impossible.

This paper shows that some controls patterns are equivalent with a composition of some data patterns. It benefits that although a workflow language implementation does not support a control pattern directly, it can support the equivalent composition of data patterns, hence in this way the self-implementations can be decreased that assists the issue of language interoperability.

Then again, all practical workflow can be constructed by a very limited number of workflow patterns. Workflow management systems usually do not support all possible patterns but a much smaller subset only. Our work points out that some of these patterns can be constructed from or converted to each other and hence, opens a way to enrich the functionality of workflow systems that support a limited set of patterns.

To demonstrate the importance of our results, we introduce use cases based on community workflows that apply data patterns to trigger control structures hence the conversibility of control- and dataflow patterns is a practically relevant issue. For instance we can use these recipes we can make a data-driven workflow management system to . As WS-PGRADE/gUSE workflow management system[1][2] became a commonly used general framework for workflow development by various scientific communities, the use cases are created using this framework to facilitate adapting our solutions in other workflows.

The paper is organized as it follows. Section II shows a brief description about how the patterns has been utilized for design a workflow language. Section III and section IV investigate two particular control patterns and offer alternative compositions of data patterns for replacing them, prove their equivalence and give detailed use cases, where the data patterns are used instead of control patterns. We summarize the results and briefly describe our further work in Conclusion and Future Work. Acknowledgment closes the paper.

II. Related work

Workflow Patterns Initiative [3] keeps track of scientific papers citing their work (till 2009 but there are much more) showing that workflow patterns became a widely accepted way for designing and for re-factoring them. Taverna used in [7] to investigate solutions for parallelism and pipeline processing, others such as YAWL(Yet Another Workflow Language) or [9] are specified directly using workflow patterns. Patterns play roles in [11] to set up a taxonomy for workflow structures, while in [14] they were used as reference models to compare existing workflow languages. Nevertheless in each case these patterns were used as they are, the relationship between particular workflow patterns was not investigated.

Van der Aalst et al. in [12] give a unified framework for combined data flow and control flow validation but they are focusing on correctness and failure aspects, not on equivalence of compositions of control and data patterns

Based on the fact that data flows and control flows are not independent a model was introduced in [13] in where one can define a workflow from both aspects. However, it does not take into count the patterns.

To summarize, field of data flows and control flows is a highly relevant and intensively investigated part of distributed process management. The relationship of data and control flow from validation point of view has been investigated but the equivalence of composition of patterns and to achieve language interoperability is still an interesting question.

III. Composing Deferred Choice pattern

In this section we introduce the if-then-else structure as it is known in programming languages. Then we introduce Deferred Choice control pattern composing with Simple Merge to express if-then-else patterns for control-driven workflow systems. After we show Task Preconditioned Data Value data pattern and prove that this pattern can express Deferred Choice, but for data-driven languages. Finally, we construct a mixed pattern from Task Precondition and Simple Merge to express if-then-else structure.

A. *If-Then-Else structure*

In programming languages beside of other basic structures such as 'sequence' and 'loop', if-then-else is the most common control structure. As is shown in Fig. 1, A can be executed on the spot, then condition Cond is evaluated, and according to its value (true or false) one particular statement (B_t if the evaluation returned true, B_f otherwise) is executed. Next, after B_t or B_f is finished, statement C is executed independently from the previous statement.

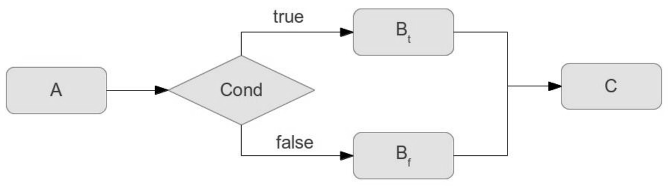


Fig. 1. if-then-else structure

During the investigation of workflow patterns, the statements are represented by nodes.

B. *Deferred Choice*

Deferred Choice pattern implements a specific split operation on one execution line to many branches of nodes. Before executing any branches (considering If-Then-Else structure it means the statement A has already been executed), one particular branch is selected according to an external decision, others will be withdrawn (condition Cond is evaluated, one of B_t and B_f can be executed). Therefore using this pattern the workflow instance will contain several disabled or "dead" branches in interpretation time, and in design time we cannot predict which branches will be interpreted and which will be excluded, since the selection is based on the environment, on the input data itself and on the user. This aspect must be taken account when, after interpreting this pattern, the next node is being evaluated. A pattern called Simple Merge is capable to deal with disabled branches of previous nodes (by taking into account which branch is disabled and not waiting for results from there), since it enables the next process if and only if one of the previous nodes is processed correctly. Therefore it is ideal to represent the interpretation of node C in If-Then-Else composition.

C. *Task Precondition - Data Value*

After a throughgoing investigation of the available workflow languages focusing data-flow concept, N. Russell et. al[5] identify several data patterns. Among others, "Task Precondition - Data Value" (hereafter we call it simply Task Precondition) defines restrictions for task execution depending on the data value. It means that a task can be enabled if and only if the precondition is satisfied.

D. *Analysis*

Comparing Deferred Choice and Task Precondition, as it is shown in Fig. 2, the difference between them is obvious.

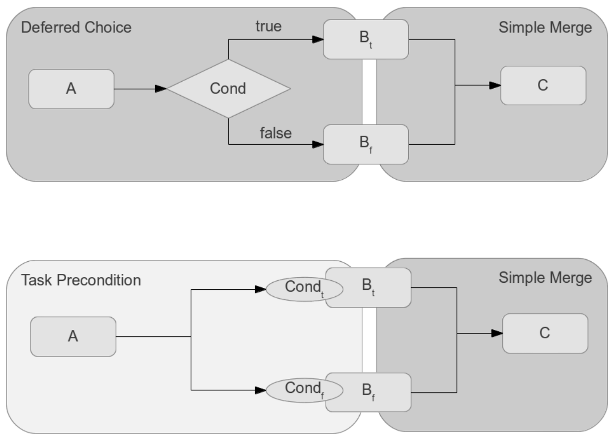


Fig. 2. Pattern compositions fitting to if-then-else structure

While Deferred Choice has a particular point in where the interpreter evaluates the condition (denoted by Cond) and enables a node depending to result of the evaluation, Task Precondition pattern evaluates independent conditions right before both subsequent nodes (B_t and B_f). Hence Task Precondition pattern does not guarantee that the true-sets of the conditions establish disjunctive sets. This tiny distinction can be resolved easily. Let us assume that Cond contains condition named C. Then we can construct the same structure for Task Precondition as we have in Deferred Choice pattern with the following preconditions: $Cond_t := C$ and $Cond_f := \neg C$. This solution effects the same structure and conditions with disjunctive result-set, therefore a Task Precondition pattern which is equivalent to a Deferred Choice.

E. Use Case - AgroHarvest

As the use case detailed below is quite complex, at first we show the pure workflow structure (see Fig. 3) that implements Data Patterns developed in WS-PGRADE/gUSE system.

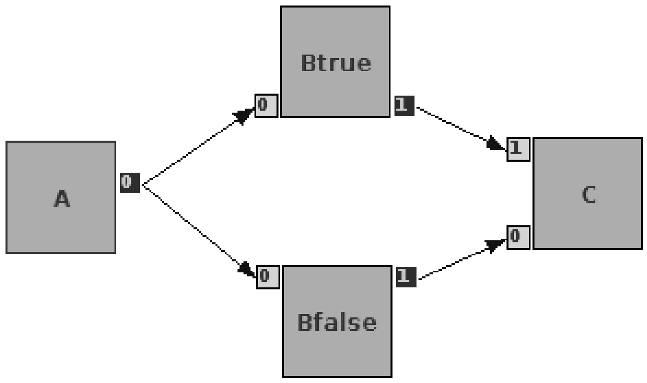
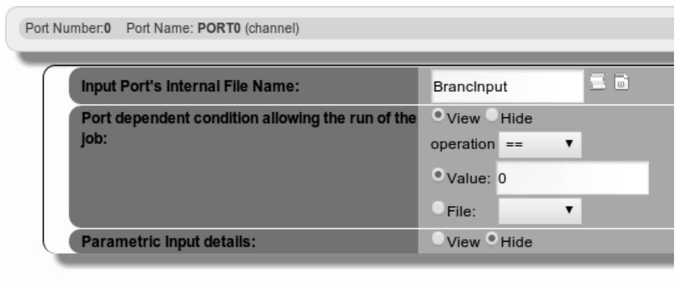


Fig. 3. Task Precondition in WS-PGRADE/gUSE

First, node A can be submitted. Depending to its process, it can generate one or more outputs associated its output box denoted by 0 (output ports are). Then, at embranchment although both $Btrue$ and $Bfalse$ nodes are enabled, data specific preconditions are set for each input port (input ports are represented by lighter grey short boxes). As it is shown in Fig. 4, data preconditions , called "Port dependent condition" property in WS-PGRADE/gUSE can be set by setting an operation (it can be $=$, $\neq$ and $contains$) and adding a particular value or a file. Evaluating a condition means a comparison of the value (or the file content) and the input file arrived to this port according to the operation. To guarantee disjunction of conditions, we must set the same condition value, but with $notequals$ operation to $Bfalse$ node's port.

According to the evaluation of the conditions, the nodes are set to disabled, or are left in enabled state. Enabled nodes are submitted then, while disabled ones are labelled with "No Input" message.

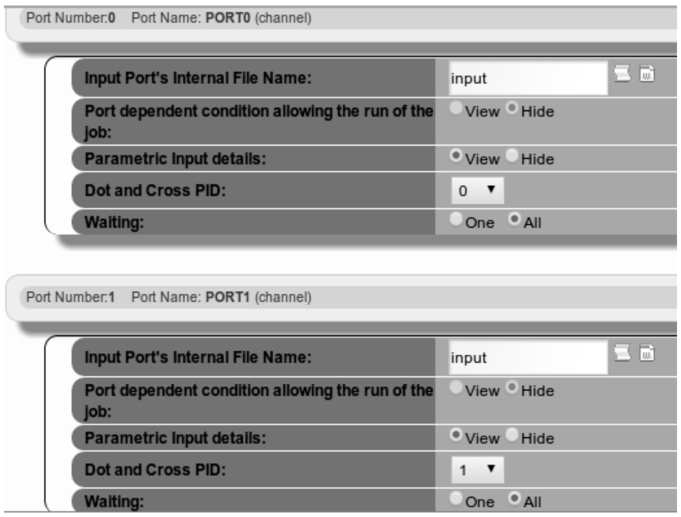


Port Number:0 Port Name: PORT0 (channel)

Input Port's Internal File Name:	BrancInput
Port dependent condition allowing the run of the job:	[✓ View] Hide operation == [✓ Value:] 0 File:
Parametric Input details:	View [✓ Hide]

Fig. 4. Configuring preconditions

Then we must apply Simple Merge pattern to implement a XOR-join, to enable node B either $Btrue$ or $Bfalse$ was submitted (and the other is disabled). Fig. 5 shows how to apply this pattern in WS-PGRADE/gUSE using the configuration panel of node B. The key point, as we do not know in design-time which branch will be executed and which will be disabled, is to set the same internal file name for each input port.



Port Number:0 Port Name: PORT0 (channel)

Input Port's Internal File Name:	input
Port dependent condition allowing the run of the job:	View [✓ Hide]
Parametric Input details:	[✓ View] Hide
Dot and Cross PID:	0
Waiting:	One [✓ All]

Port Number:1 Port Name: PORT1 (channel)

Input Port's Internal File Name:	input
Port dependent condition allowing the run of the job:	View [✓ Hide]
Parametric Input details:	[✓ View] Hide
Dot and Cross PID:	1
Waiting:	One [✓ All]

Fig. 5. Configuring Simple Merge pattern

Then we need to set collector property on both ports ("Waiting" row in the Figure is set to "All") to allow to enable node B in the case as well, when it has data dependencies with disabled nodes only.

The solution is applicable for n generated outputs. In this case the quantity of $Btrue$ node's true-set is x, and $n-x$ for $Bfalse$. But, because of we set the same internal file name for both input ports in node B, altogether n input files will be renamed to this internal name independently from which job (which branch) resulted it. To resolve overwriting issues, the file names will be extended by a unique number respectively.

1) about Agro-Know Harvest application: To confirm that it is necessary to support Deferred Choice structures, as a real-life application we selected Ariadne application from Agro-Know Technologies, a project partner involved in ag-Infra project (see Acknowledgment). Agro-Know is an innovative, research-oriented enterprise focusing on knowledge-intensive technology for agriculture and rural development. Mainly research of knowledge-intensive technology aims to make knowledge, the know-how enable, searchable, reusable independently from location, format or scope. In this sense beside the importance of the information stored as data at least much important the information about the data itself, namely the metadata. In large scale metadata informations are stored in datastores, and, of course so many datastores are available around the world in the field of agriculture. Agro-Know defines workflows to harvest, manage metadata and make it interoperable among different formats.

Ariadne process operates on metadata datastores. Once a datastore is validated (there is a step that checks its validity) the stored metadata can be harvested, otherwise the workflow stops and user must be notified about the failure. Harvesting means downloading all metadata stored in the datastore. Then all of the metadata is validated one by one against a particular format (e.q. OAI-LOM), if a metadata is invalid, it must be converted as a next step. All the valid and the converted metadata are stored in remote storages in a quickly searchable folder hierarchy.

Clearly this workflow contains some points, e.g. validating a metadata, that makes it be ideal to illustrate Deferred Choice pattern.

The workflow shown in Fig.6 implements Ariadne process. The following list details the functions of its nodes.

- GenT: Splits a list of datastore URIs, and passes with the set of format respectively, in which the metadata of a particular datastore must be stored.
- Target: it validates each datastore target against the given schema.
- Registrate: registrates the datastore in the remote storage, if the target is valid.
- GetR: gets the results of the registration.
- Harvest: gets all metadata from a particular datastore target, stores them as local files. It runs in as many instances concurrently as many datastores splitted by GenT.
- OAIVal: Validates a metadata against OAI_LOM format.
- ItsValid: Dummy node for valid metadata.
- OAITrans: Converts a metadata to the requested format if and only if the validation fails.
- Upload: uploads metadata to the remote storage.
- Regis: registers each dataset associated to their target registered before.

In aspect of Deferred Choice patterns, the interesting parts of the workflow are enclosed by black circles. Within normal lined circle the process is the following. Both nodes $ItsValid$ and $OAITrans$ have disjunctive preconditions, hence according to the validity of a given metadata (this information comes from node $OAIVal$ as output) either $ItsValid$ or $OAITrans$ will be submitted, but never both. Nevertheless nodes within the dashed circle do not fit directly to the pattern since the datastore must be registered and harvested in the same case, when the datastore is valid. In other words the true-sets of the preconditions adjusted to node $Regis$ and $Harvest$ are not disjunctive.

IV. Composing Multiple Instances with multiple indexes

A. Multiple instances with a priori Run-Time Knowledge

This pattern(MultiInstance in short) defines that one node created at design-time can generate several node instances at run-time depending on various conditions such as resource availability or data. Since during the interpretation of the workflow these conditions can change and can be evaluated time-by-time, the number of instances created is known only right before interpreting the node definitely. All instances are independent and they are submitted and executed concurrently. Then the branches may be synchronized at completion if it is necessary. Let us assume a workflow containing 3 nodes, A, PS and B connected respectively. Run-time view of this

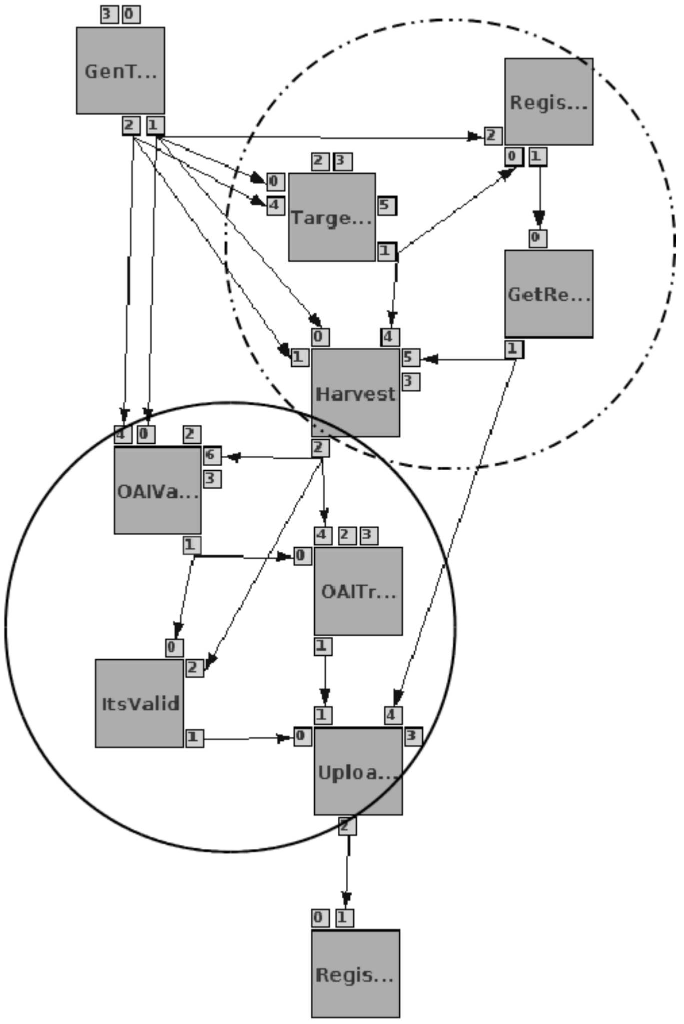


Fig. 6. Harvester Workflow

pattern is shown in Fig. 7. after processing node A, PS_1, PS_2, PS_3 are going to be created, none the less only one, PS was defined at design-time.

Although it has not been described explicitly in [4], we assume that all nodes contains by patterns can be replaced by an other, or the same pattern. In this scenario we assume to ha 2 workflows, both contains 3 nodes, A PS and B, connected respectively, and both PS nodes implements MultiInstance pattern. Let us denote the first workflow by index 0 and the second by index 1, all the nodes are labeled with these indexes. Then we embed workflow 2 to the first's middle node (PS_0). Fig. 8 illustrates this structure during enactment. After processing A_0, the subsequent node is going to be replaced by n copy of it. Then, as this node not a real node, just a container to an other workflow, the other workflow will be enacted in as many copy as many container nodes we have. The enactment is done separately and parallel and shown as A_1, PS_{11}...$B1$ nodes in the Fig. 8. As the embedded workflow implements MultiInstance pattern as well, processing its middle node means processing n copies of it in parallel. Finally the last node (B_0)is going to be processed.

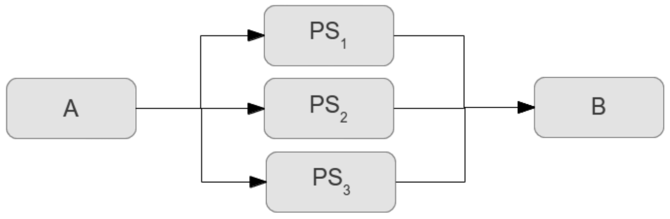


Fig. 7. Simple Multiple Instances pattern

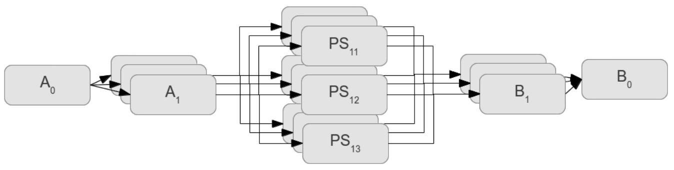


Fig. 8. Recursive Multiple Instances pattern

B. Multiple Instance Task Data Pattern

While MultiInstance pattern introduced in the previous paragraph implements the ability to create more instances from one node, this pattern focuses on the data aspects of this issue. According to how the data can be passed to the multiple instances, this pattern contains three sub-patterns:

1) Instance Specific Data Passed by Value
2) Instance Specific Data Passed by Reference
3) Shared Data Passed by Reference pattern

While the first one passes the data as is to the multiple instances, the others just passes the reference of the data. Other difference is while first and second works with instance specific data allowing to pass different data for each instance, the third do not allow it, this patterns works with shared data only. In all case the data passing occurs when the multiple instance task is enabled. Since our work does not take into account these aspects, any of them can be chosen. In the followings by mentioning MultiDataInstance pattern will mean the Multiple Instance Task - Instance Specific Data Passed by Value pattern. This pattern is illustrated by the following figure.

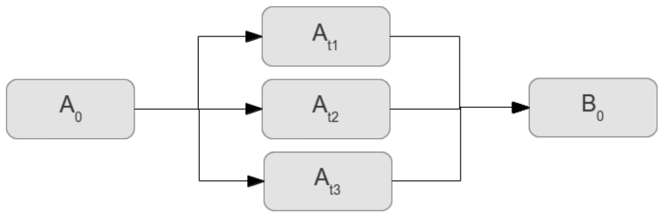


Fig. 9. Data Interaction to Multiple Instance Task - Instance Specific Data Passed by Value pattern

C. Sub-Workflow Decomposition

Sub-Workflow Decomposition pattern deals with data aspects of embedding a workflow into a node. It allows that any data, that is accessible by the node may be passed to the workflow embedded and vice versa, all output generated during the interpretation of the embedded workflow can be passed back to the upper level of execution. Usually this pattern is implemented to allow embedding in concept of black-boxes, namely the embedded workflow has one particular entry and exit points, during its interpretation all outputs generated are inaccessible from the external nodes. The concept of this pattern is shown in Fig. 10.

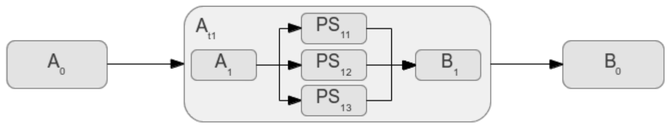


Fig. 10. Data Interaction - Block Task to Sub-Workflow Decomposition - Explicit Data Passing via Data Channels pattern

To summarize we can say that the simple MultiInstance pattern and MultiDataInstance patterns are equivalent (disregarding that MultiInstance as a Control Pattern can be affected by several circumstances, while process of MultiDataInstance as a Data Pattern is restricted to the availability of the data), but as we shown in Fig. 8, MultiInstance pattern can be applied recursively, while MultiDataInstance not. Nevertheless in composition with Sub-Workflow Decomposition we can achieve the same functionality.

D. Analysis

After introducing the control and data pattern components, we prove their equivalence using formal description of graphs.

Let us define concept of Double Tree as follows:

Definition 1: $DT := (D, p)$, where A denotes the set of nodes, and p denotes set of pairs or nodes representing arcs.

$D := \{s, t, \{A^0...A^{n-1}\}\}$ and

$\forall a \in A^0 :!\exists(s, a) \in p$

$\forall a \in A^i \wedge a' \in A^{i+1} :!\exists(a, a') \in p(i \in [0, n-2])$

$\forall a \in A^{n-1} :!\exists(a, t) \in p$

and $\forall i \in [1..n] : |A^i| = |A^{n-i}|$

Technically it means that Double Trees have one entry and one exit points, and between them levels are defined (A^i) according to the arcs among the nodes. The last restriction says that the number of the nodes and their degrees are symmetric to the medial level of nodes.

It can be seen clearly that none but those structures satisfy this definition which are applied by Multiple Instance pattern. Then we have a constructive definition in a point of view, which is familiar with the Data Flow constructions.

Let us define an initial graph with only three nodes connected respectively. Then repeat the following function against this graph in any number of times. Every graph constructed in each iteration will be applied by definition.

Definition 2: $size(G) := |A^{n/2}|$

Definition 3: $BLOW : (DT \times DT_N) \rightarrow DT$, where both G and G_N denote double trees with n=1, but while size of G is not restricted, size of G_N is denoted by N in its index. It follows that each graphs has only one For simplicity we use their detailed definition in the followings (D,p).

Initially $A = A_1, p = p_1$

$BLOW((D_1, p_1), (D_2, p_2)) := \{(D, p) | \forall a_i \in A_1^{n/2} :$

$D = D \bigcup D_2 \setminus a_i \wedge$

$p = p \bigcup p_2 \bigcup (s_2, s_1) \bigcup (p_1, p_2) \setminus (s_1, a_i) \setminus (a_i, p_1)\}$

Theorem 1: function blow is idempotent in Double Trees.

Proof: As it is defined, function $BLOW$ replaces nodes in the middle level by a complete complete graph(DT_N) given as second parameter. Since DT_N has only one level beside its entry and exit points, the result graph will contain odd number of the levels. Therefore its middle level can be identified obviously, which means that the function can be applied again. As DT_N trivially symmetric on its middle level. None the less $|A^{n/2}|$ nodes are being removed for DT, but the same number of entry and exit points are being included and connected accordingly and then, as a new middle level, A^0 of DT_N, which has N nodes, will be included and connected as well $|A^{n/2}|$ times, so the new middle level will contain N*$|A^{n/2}|$ nodes.

As Theorem 1 results that a Double Graph can be derived from two others by function blow, and the function can be used iteratively. As its process precisely follows the way described by Data Flow patterns, and generates the same set of Double Graphs, it can be taken cognizance of the equivalence between the composition of specified Data Flow patterns and the Multiple Instance Control Flow pattern.

E. Use case - Sea Vessel Tracking Application

Correlation Systems provides solutions to process and analyse large sets of geospatial data. They focus on the integration of open source and geospatial data analysis and implementation of real time resource allocation method. Correlation Systems targeted the marine security community developed WWAIS Vessel Tracker application. WWAIS first, processes partial event data received from sea vessels Accounting Information System (AIS) transmitters. Next, it extrapolates this data in order to define vessels activities. Finally, this data is compared to trends of activities along world sea-trade routes to be able to define the vessel route and identify any unexpected activity. The extrapolation process is an essential part of analysing vessels activities and routes at sea since AIS receivers are placed in major cities only. As a result, sufficient data can be collected, if the vessel runs next to them, and little or no data when vessels far out at sea. The extrapolation process is very CPU and time intensive because of three reasons. At first, a large amount of vessel data should be processed. Currently data is available for more than 200.000 vessels. Secondly, the size of the geographical data is huge. In order to accurately map a voyage its route must be plotted based on coastline data. A detailed world coastline map can contain many complex polygons consisting of several million coordinate points. Thirdly, a Dijkstras variation compares each ship event to world shipping route data. The matrix of world shipping route data contains more than 2 million coordinate points. Considering that each vessel and its voyages can be processed separately, this application is an ideal candidate for parallel processing.

The WWAIS Vessel Tracker application is implemented as a number of RESTful web services running on several Tomcat instances. There are web services that prepare the raw route-data, others analyse the route (different algorithms can be implemented and used) while further ones finalize and display the route. Correlation Systems created a workflow which coordinates the route calculation using web services as black-box applications. In the workflow two processing algorithms should be executed in parallel to analyse the route using different models. Outputs of these algorithms are compared and the best results are selected. Analysing a complete voyage means splitting it to different routes, executing these two algorithms to calculate each route and finally collecting and comparing routes to merge and create the voyage history. Basically each route analysis can be represented as a parameter sweep step. Similarly the analysis of a set of vessels follows the same technique but in a larger scale. First, a set of vessels must be identified, next, the voyage analysis must be processed on each vessel and finally results of all vessels should be collected and evaluated. Thus, the WWAIS application is a parameter sweep application that contains another parameter sweep application, which consists of some external web service callings. The use case is defined in WS-PGRADE/gUSE workflow system. Since it must be guaranteed that the routes are assigned to groups according to their vessel ID, parameter sweep executions must be implemented in two levels by defining Multiple Instance Task Data Pattern(MITDP in short) and the internal (called Embedded) and the external (called Master) workflows and connecting them by Sub-Workflow Decomposition pattern (SWD in short). The gUSE workflow on the Fig 11 shows the workflow structure that covers the two parameter sweep parts: Master and Embedded workflows. The Master workflow coordinates the voyage calculation of all vessels as a high-level MITDP. This workflow contains a generator job (VesGen job) and collector job (VesCol job) to manage the Embedded workflow instances. The generator jobs input is the set of vessel IDs. The Embedded workflow (represented by the Emb job in the Master workflow) calculates and analyses the voyage of each vessel as a low-level MITDP. This job is executed in parallel by as many instances as many vessel IDs are in the ID set. In the figure the dashed line represents the mapping of the inputs among Master and Embedded workflow, what clearly used for define SWD among them. In the Embedded workflow the first job (VoyGen job) splits the vessels voyage into routes and saves the vessels ID for each route in the configuration file. Next, the interpreter creates as many instances of the Corr job as many routes the VoyGen job generated. The Corr instances are run in parallel. Finally VoyColl collects and frames all the corrected sub-routes together.

V. Conclusion and Future Work

In this paper after introducing different data and control patterns we created compositions from them and we proved their equivalence. Then we showed their interpretation on use case community workflows developed in WS-PGRADE/gUSE system. For further work we are planning to investigate more data and control pattern compositions and their relationships, especially for complex control patterns. Decomposing these patterns, and finding analogous data patterns for each component can lead us to new control-inspired complex data patterns. These complex patterns could extend the opportunities of pure data flow systems.

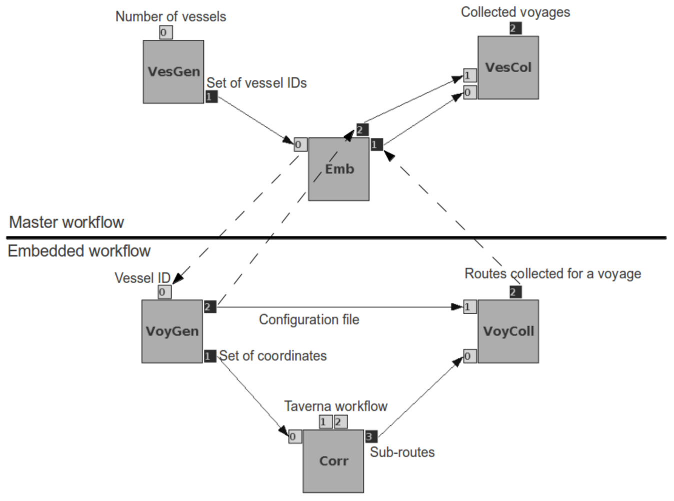


Fig. 11. WWAIS Workflow

ACKNOWLEDGMENT

The authors would like to thank to Correlation Systems, which delivered WWAIS application for us to have a real-life use case for Deferred Choice pattern.

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 283481 (SCI-BUS) and under grant agreement no 283770(agINFRA).

REFERENCES

[1] P. Kacsuk et. al.: *WS-PGRADE/gUSE Generic DCI Gateway Framework for a Large Variety of User Communities* Journal of Grid Computing, 9, 4, 479-499, 2012

[2] P. Kacsuk et. al.:*P-GRADE portal family for Grid infrastructures* Concurrency and Computation: Practice and Experience journal, 23, 3, 235-245, 2011

[3] www.workflowpatterns.com [accessed 27. March 2013]

[4] N. Russell et. al.:*Workflow Control-Flow Patterns: A Revised View.* BPM Center Report BPM-06-22 , BPMcenter.org, 2006.

[5] N. Russell et. al.: *Workflow Data Patterns* QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.

[6] Oinn, T. et. al.: *Taverna: a tool for the composition and enactment of bioinformatics workflows*, Bioinformatics, 20(17), 3045-3054.,2004.

[7] P. Missier, et.al. *Taverna, reloaded* Scientific and Statistical Database Management, 471-481, 2010.

[8] J. Mendling, et. al. *EPC markup language (EPML): an XML-based interchange format for event-driven process chains (EPC).* Information Systems and e-Business Management, 4(3), 245-263. 2006

[9] T. McPhillips et al. *Scientific workflow design for mere mortals* Future Generation Computer Systems - The International Journal of Grid Computing - Theory Methods and Applications, 25(5): 541-551. 2009

[10] R. Lucchi et. al. *A pi-calculus based semantics for WS-BPEL* The Journal of logic and algebraic programming, 70(1), 96-118. 2007

[11] Yu, J., Buyya, R. *A taxonomy of workflow management systems for grid computing* Journal of Grid Computing, 3, 3-4, 171-200. 2005

[12] N. Trcka et. al.: *Analyzing control-flow and data-flow in workflow processes in a unified way* Computer Science Report, (08-31). 2008

[13] S. Fan *Dual workflow nets: Mixed control/data-flow representation for workflow modeling and verification*, Advances in Web and Network Technologies, and Information Management,433-444 2007

[14] A. Shiroor *Scientific workflow management systems and workflow patterns* International Journal of Business Process Integration and Management, 5(1), 63-78.2010