Towards formal model for location aware workflows

Preprint version of "Doriana Medic and Marco Aldinucci. Towards formal model for location aware workflows", 2023 IEEE 47th Annual Computers, Software, and Applications Conference, doi: 10.1109/COMPSAC57700.2023.00289

Towards formal model for location aware workflows

Doriana Medić
Department of Computer Science
University of Turin
Turin, Italy
doriana.medic@unito.it

Marco Aldinucci
Department of Computer Science
University of Turin
Turin, Italy
marco.aldinucci@unito.it

***Abstract*—Designing complex applications and executing them on large-scale topologies of heterogeneous architectures is becoming increasingly crucial in many scientific domains. As a result, diverse workflow modelling paradigms are developed, most of them with no formalisation provided. In these circumstances, comparing two different models or switching from one system to the other becomes a hard nut to crack.**

This paper investigates the capability of process algebra to model a location aware workflow system. Distributed π-calculus is considered as the base of the formal model due to its ability to describe the communicating components that change their structure as an outcome of the communication. Later, it is discussed how the base model could be extended or modified to capture different features of location aware workflow system.

The intention of this paper is to highlight the fact that due to its flexibility, π-calculus, could be a good candidate to represent the behavioural perspective of the workflow system.

***Index Terms*—Workflow, distributed, π-calculus, locations**

I. Introduction

Nowadays, designing complex applications and executing them on large-scale topologies of heterogeneous architectures is becoming increasingly crucial in many scientific domains. Along with it, the workflow modelling paradigms have evolved in different directions to adapt to the needs of different users, focusing on generality and user-friendliness or favouring flexibility at the expense of greater complexity. As a consequence, there are more than three hundred different Workflow Management Systems [8] and comparing their features or migrating from one system to the other is becoming a burden.

One of the reasons for mentioned complexity is the fact that WMS implementations rarely come with formalised concurrent execution semantics. Some valuable efforts have been made to develop unifying interfaces for the workflow coordination model, e.g. the Workflow Patterns initiative [9] or the Common Workflow Language [10] open standard. They encompass concurrent executions, even if their semantics is not fully formalised.

In the literature, up to our knowledge, there are few workflow management systems for which formal models have been developed: Taverna [1], [15], where workflow language is defined in functional terms using the computational lambda calculus [16]; Kepler [2] based on Process Networks [3] and BPEL [4] giving a complete formalization of all control-flow constructs and communication actions of BPEL in Petri nets. A detailed representation of the workflow patterns has been made with YAWL [5], a workflow language built upon experiences with the languages supported by contemporary workflow management systems. The base model of this language are Petri nets because of their power in dealing with the specification of control-flow dependencies in workflows. YAWL provides the representation of all workflow patterns, as well as tool support and interfacing with various workflow tools.

Different attempts to describe the behaviour of the workflows by process theory models can be found in the literature. In [11], the π-calculus [18], a well-established process algebra, is claimed to be a natural fit for modelling a workflow system. The processes able to dynamically change their structure can be naturally expressed by π-calculus thanks to its feature called *mobility* (the capability of sending names along the channels where received names can be used as channel names in future communications). Discussion about describing a workflow system by π-calculus or Petri nets is given in [12]. Both paradigms have their advantages and disadvantages and should be used with slightly different aims, leading to more theoretical and compositional, or functional and user-friendly models.

The first step in the analysis of the capabilities of the π-calculus with respect to workflow patterns [13] is given in [6]. It introduces a class of workflow patterns formalised with concise and unambiguous expressions. The behaviour of each workflow pattern has been precisely defined thanks to the execution semantics of the π-calculus. Another approach to the use of the π-calculus for workflow definitions is given in [17] and it concentrates on basic control flow constructs and the definition of activities. Besides π-calculus, in [19], to model Web Service Choreography Interface descriptions, Calculus of Communication Systems (CCS) [20] is used. It is shown that the obtained formal model can be exploited for the definition of compatibility and replaceability tests between Web services, and the automatic generation of adaptors that can bridge the differences between a priori incompatible Web services.

This paper is following a research line of [6] and aims to provide a broader exploitation of the process theory for

This work has been partially supported by the Spoke 1 "FutureHPC & BigData" of ICSC - Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing, funded by European Union - NextGenerationEU. This research is also partly supported by project EUPEX, which has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101033975.

describing workflow systems. Differently from [6], where the highlight is on giving the formalisation for all workflow patterns, the purpose of this work is to explore the features required by location aware workflows with the intention to strengthen the idea of having a formal representation of a workflow model given with a distributed process algebra. To do that, a distributed π-calculus [7] is chosen as a process model paradigm and used as a base for the workflow description. Distributed π-calculus is based on the π-calculus, to which a network layer and a primitive migration construct are added. It is a compositional process algebra built by processes and channels through which processes communicate. The flexibility of distributed π-calculus allows for further customisation of the proposed description of a workflow system, based on different workflow and location specifications. One of the advantages of π-calculus is that it is equipped with the equivalence theory. Two processes can be checked for equivalence in terms of bisimulation [22], what is particularly useful in business process optimization.

The paper is structured as follows. Section II gives an interpretation of the considered workflow setting and provides the syntax of the distributed π-calculus, tailored to described workflow. The operational semantics is presented in Section III, while Section IV contains a discussion on the given formalisation suggesting different variations of it. Section V concludes the paper.

II. Formalising location aware workflows

This paper considers an abstract definition of the location aware workflows leaving a lot of freedom for the implementation of different specifications. Directed Acyclic Graph (DAG) is chosen for the workflow representation. Nodes of the graph stands for the workflow steps to be executed, while annotated directed edges (arrows) are defining the flow of the workflow execution (dependencies between the steps) and the name of the port (channel) through which two steps are connected. Each step is bounded to the location in which is to be executed. It is allowed for one step to be executed on more locations[1] and for more steps to be mapped to one location, in which case the steps can be executed in parallel if there is no dependency between them and resource allows for it. It is assumed that the initial data is already positioned on the first location of the workflow (location to which the first step is mapped to) and it is abstracted from the actual internal computation of a step and the data type used for it. Denotation of the workflow components is done in the following way: steps are represented with $S_1, S_2, \ldots$, channel names with $a, b, c, \ldots$, data to be transferred with $d_1, d_2, \ldots$ and locations with $l_1, l_2, \ldots$. To make it straightforward, each step, channel, location and data produced as a result of a step execution are unique (even if there could be the steps which during the execution do not change received data).

[1]Mapping step to more locations and not just dividing original step into more steps is used in the case when, for instance, there is the same code executed on multiple computing elements, like MPI (Message Passing Interface) [21] program.

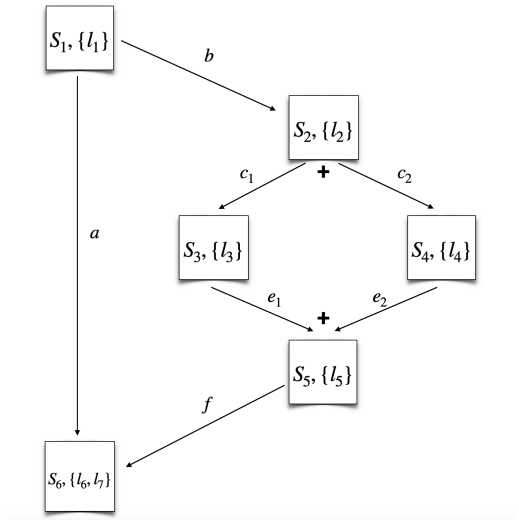

Fig. 1. Example of a workflow graph

As an example, Figure 1 depicts a workflow which shall be used through the paper to give an idea how formal model captures different operations (not as the representation of the actual workflow application). Workflow DAG consists of the six nodes (six steps) represented with squares and denoted with the couple S, L containing information about the step and the locations to which the step is bounded (set of locations L). The edges are annotated with the name of the channel. Step S_1, after its internal computation, is sending data to the steps S_2 and S_6 mapped to the corresponding locations (i.e. from location l_1 data is sent to locations l_2, l_6 and l_7), while the step S_2 is sending its data to the step S_3 or to the step S_4. The workflow is completed when step S_6 is executed. The behaviour of each step except the initial and the last steps is: receiving data to be processed, processing data and sending resulting data to the next steps. In this example, step S_6 is bounded to two locations l_6 and l_7 while the rest of the steps are executed each on a single location.

A distributed π-calculus [7] tailored to our setting together with the examples explaining how to model the workflow of Figure 1 are presented in the following. The representation of the workflow patterns by π-calculus, without considering locations, is done in [6]. This work is following the approach of [6] with a difference that distributed π-calculus [7] is used and locations are added.

Distributed π-calculus systems [7], denoted with M, adapted to our setting, are given with the following syntax:

$$M ::= l[\![P]\!] \mid (M \mid M) \mid \mathbf{0}$$

The parallel composition of systems is denoted with $(M \mid M)$, while the idle system is represented by $\mathbf{0}$. Term $l[\![P]\!]$ defines a process P to be performed on the location l. In the context of the workflow, process P can be seen as a step S to be executed on the location l. The π-calculus process P is defined as:

$$P ::= \pi.P \mid (P \mid P) \mid P + P \mid \texttt{stop}$$

where the parallel composition and the choice operator are represented with terms $P \mid P$ and $P + P$, respectively, while the process termination is denoted by `stop`. Process $\pi.P$ stands for a sequential execution i.e. the process P can be executed only after the prefix π is performed. Prefix denotes a simple action to be executed and it is defined with:

$$\pi ::= \overline{a}\langle d\rangle \mid a(x) \mid \texttt{goto}\ l \mid \tau_S$$

Term $\overline{a}\langle b\rangle$ depicts sending the data d over the channel named a, while $a(x)$ stands for the process able to receive the certain data over the channel a. The received data is then bounded to the placeholder x. The internal computation of a step is indicated with τ_S.

In distributed π-calculus, to be able to communicate with a process on a different location it is necessary to explicitly move the data to the targeting location and then do the communication. This is enabled with the prefix $\texttt{goto}\ l$, indicating to move to the location l (having $\texttt{goto}\ l.P$ would imply to move the process P on the location l).

In the workflow setting, the data is transferred from one location to the other through the corresponding channel and then used for the internal computation of the step. In the formal description, this behavior is modelled by first moving to the targeting location with action $\texttt{goto}\ l$ and then communicating through the corresponding channel. The workflow model is not changing, while the execution plan would have one additional operation every time when it is necessary to move data from one location to the other.

In the following example, it is illustrated how workflow given in DAG can be syntactically represented with the calculus defined above.

Example 1: The formal syntactic representation of the workflow DAG given on Figure 1, can be done with the following steps:

- Nodes S_i, L_i when $i = 1, \ldots, 6$, can be seen as systems M_i, such that

$$M_i = \begin{cases} l_i[\![S_i]\!] & \text{for } L_i = \{l_i\} \\ l_1[\![S_1]\!] \mid \ldots \mid l_n[\![S_1]\!] & \text{for } L_i = \{l_1, \ldots, l_n\} \end{cases}$$

  In words, if step is mapped to $n > 1$ locations then the system M_i shall contain n subsystems composed in parallel. Considering graph on Figure 1, for instance, the node $S_1, \{l_1\}$ becomes the system $M_1 = l_1[\![S_1]\!]$, while the node $S_6, \{l_6, l_7\}$ translates into system $M_6 = l_6[\![S_6]\!] \mid l_7[\![S_6]\!]$.
- The entire workflow, denoted with W can be defined as a parallel composition of systems M_i:

$$W = l_1[\![S_1]\!] \mid \ldots \mid l_5[\![S_5]\!] \mid l_6[\![S_6]\!] \mid l_7[\![S_6]\!]$$

- Step S_1 (already containing data) is defined as

$$S_1 = \tau_{S_1}.(\texttt{goto}\ l_2.\overline{b}\langle d_1\rangle \mid \texttt{goto}\ l_6.\overline{a}\langle d_2\rangle \mid \texttt{goto}\ l_7.\overline{a}\langle d_2\rangle)$$

  The internal computation of the step S_1 (denoted with τ_{S_1}) is followed by moving to the locations l_2, l_6, l_7 (depicted with the $\texttt{goto}\ l$ prefix) and executing the corresponding communications.

$$\text{(OUT)} \quad \overline{a}\langle d\rangle \xrightarrow{\overline{a}\langle d\rangle} stop$$

$$\text{(IN)} \quad a(x).P \xrightarrow{a(d)} P\{d/x\}$$

Fig. 2. (OUT) and (IN) rules

- Step S_2 is modelled as

$$S_2 = b(x).\tau_{S_2}.(\texttt{goto}\ l_3.\overline{c_1}\langle d_3\rangle + \texttt{goto}\ l_4.\overline{c_2}\langle d_3\rangle)$$

  Once data is received over the channel b, step S_2 executes its internal computation and sends resulting data d_3 through the channel c_1 to the step S_3 or the channel c_2 to S_4.
- Steps S_3 i S_4 are modelled in a similar way:

$$S_3 = c_1(y).\tau_{S_3}.\texttt{goto}\ l_5.\overline{e_1}\langle d_4\rangle$$
$$S_4 = c_2(z).\tau_{S_4}.\texttt{goto}\ l_5.\overline{e_2}\langle d_5\rangle$$

  Both steps receive data, process it, produce a new one and output it over the channels e_1 and e_2.
- The internal computation of the step S_5 is triggered by the data received over the channel e_1 or e_2. The resulting data is sent over channel f to the locations l_6 and l_7 to which step S_6 is mapped to:

$$S_5 = e_1(x).\tau_{S_5}.(\texttt{goto}\ l_6.\overline{f}\langle d_6\rangle \mid \texttt{goto}\ l_7.\overline{f}\langle d_6\rangle) + e_2(y).\tau_{S_5}.(\texttt{goto}\ l_6.\overline{f}\langle d_6\rangle \mid \texttt{goto}\ l_7.\overline{f}\langle d_6\rangle)$$

- The last step, S_6, can perform the internal computation only when all data is received. Therefore, it can be represented as

$$S_6 = a(x).f(y).\tau_{S_6} + f(y).a(x).\tau_{S_6}$$

The algorithm described above, illustrates how to syntactically represent the workflow DAG with the distributed π-calculus. In the following, each operation of the syntax is paired with the corresponding behaviour.

III. Operational semantics

This section provides the operational semantics of the distributed π-calculus [7], adapted to the syntax defined in Section II and the workflow setting. First, the two simple rules not involving locations are presented end depicted in Figure 2. The output rule (OUT) is describing the sending of the data d over the channel a. There is no continuation after the action is executed since asynchronous π-calculus is considered. In the input rule (IN), data d is received and bounded to the placeholder x. The computation continues with the execution of the process P where each occurrence of the variable x is substituted with d. In the workflow setting, the received data d shall be used for the internal computation of a step.

The operational semantics, involving locations, for distributed π-calculus, is given in Figure 3. The description of each rule is provided below:

- Rule (STEP) enables the internal computation of step S.

$$\text{(STEP)} \quad l[\![\tau_S.P]\!] \rightarrow l[\![P]\!]$$

$$\text{(SPLIT)} \quad l[\![P \mid P_1]\!] \rightarrow l[\![P]\!] \mid l[\![P_1]\!]$$

$$\text{(MOVE)} \quad \frac{l' \in L}{l[\![\texttt{goto}\ l'.P]\!] \rightarrow l'[\![P]\!]}$$

$$\text{(COMM)} \quad \frac{\overline{a}\langle d\rangle \xrightarrow{\overline{a}\langle d\rangle} stop \quad a(x).P \xrightarrow{a(d)} P\{d/x\}}{l[\![\overline{a}\langle d\rangle]\!] \mid l[\![a(x).P]\!] \rightarrow l[\![P\{d/x\}]\!]}$$

$$\text{(CHOICE-1)} \quad \frac{P \xrightarrow{\alpha} P',\ \alpha = \{\overline{a}\langle d\rangle, a(d)\}}{l[\![P + P_1]\!] \rightarrow l[\![P']\!]}$$

$$\text{(CHOICE-2)} \quad \frac{l[\![\texttt{goto}\ l'.P]\!] \rightarrow l'[\![P]\!]}{l[\![(\texttt{goto}\ l'.P) + P_1]\!] \rightarrow l'[\![P]\!]}$$

$$\text{(PAR)} \quad \frac{M \rightarrow M'}{M \mid M_1 \rightarrow M' \mid M_1}$$

Fig. 3. Semantics rules of a distributed π-calculus

- Rule (SPLIT) allows a system composed of the processes in parallel to be split into smaller systems which can continue computation separately. For instance, in the workflow setting, this rule is applied when the step has two output actions targeting different locations.
- Moving the step from one location to the other is defined with the rule (MOVE). Mostly, this rule is applied to move the data to the targeting location and in this way enable the communication.
- The communication is done through the rule (COMM). It can be applied only if both, input and output, actions are able to execute on the same location. To communicate with the process on a different location, one should first apply the rule (MOVE).
- The choice rules select one of the possible options of computations and discard the others. Rule (CHOICE-1) considers simple input and output actions and it can be applied only with the assumption that these actions can be executed. Similar for the rule (CHOICE-2) and action $\texttt{goto}\ l$. The reason to have two separate choice rules is only to avoid overloading a single one.
- Lastly, the parallel rule is allowing a system to be executed as a part of a larger system. One could notice that there is no direct rule executing the system $l[\![P \mid P']\!]$. This is due to the fact that the system $l[\![P \mid P']\!]$ can be split into two separate systems $l[\![P]\!]$ and $l[\![P']\!]$ on which the rule (PAR) can be applied.

For parallel, choice and communication rules of Figure 3 there are the symmetric rules which are omitted.

How the semantics defined above captures the behaviour of the workflow step S_1 of Example 1 is illustrated in the following. Since the step S_1 is the initial step, it already contains the necessary data and can perform the internal computation by applying the rule (STEP):

$$l_1[\![\tau_{S_1}.(\texttt{goto}\ l_2.\overline{b}\langle d_1\rangle \mid \texttt{goto}\ l_6.\overline{a}\langle d_2\rangle \mid \texttt{goto}\ l_7.\overline{a}\langle d_2\rangle)]\!] \rightarrow$$
$$l_1[\![\texttt{goto}\ l_2.\overline{b}\langle d_1\rangle \mid \texttt{goto}\ l_6.\overline{a}\langle d_2\rangle \mid \texttt{goto}\ l_7.\overline{a}\langle d_2\rangle]\!]$$

The next action to do is to split obtained system into the smaller ones by applying twice the rule (SPLIT):

$$l_1[\![\texttt{goto}\ l_2.\overline{b}\langle d_1\rangle \mid \texttt{goto}\ l_6.\overline{a}\langle d_2\rangle \mid \texttt{goto}\ l_7.\overline{a}\langle d_2\rangle]\!] \rightarrow\rightarrow$$
$$l_1[\![\texttt{goto}\ l_2.\overline{b}\langle d_1\rangle]\!] \mid l_1[\![\texttt{goto}\ l_6.\overline{a}\langle d_2\rangle]\!] \mid l_1[\![\texttt{goto}\ l_7.\overline{a}\langle d_2\rangle]\!]$$

Now, the rule (MOVE) can be applied to shift the process $\overline{b}\langle d_1\rangle$ into the location l_2:

$$l_1[\![\texttt{goto}\ l_2.\overline{b}\langle d_1\rangle]\!] \mid l_1[\![\texttt{goto}\ l_6.\overline{a}\langle d_2\rangle]\!] \mid l_1[\![\texttt{goto}\ l_7.\overline{a}\langle d_2\rangle]\!] \rightarrow$$
$$l_2[\![\overline{b}\langle d_1\rangle]\!] \mid l_1[\![\texttt{goto}\ l_6.\overline{a}\langle d_2\rangle]\!] \mid l_1[\![\texttt{goto}\ l_7.\overline{a}\langle d_2\rangle]\!]$$

By putting together the obtained system and the system M_2 containing the step S_2, the communication can be done through the rule (COMM):

$$l_2[\![\overline{b}\langle d_1\rangle]\!] \mid l_1[\![\texttt{goto}\ l_6.\overline{a}\langle d_2\rangle]\!] \mid l_1[\![\texttt{goto}\ l_7.\overline{a}\langle d_2\rangle]\!] \mid$$
$$l_2[\![b(x).\tau_{S_2}.(\texttt{goto}\ l_3.\overline{c_1}\langle d_3\rangle + \texttt{goto}\ l_4.\overline{c_2}\langle d_3\rangle)]\!] \rightarrow$$
$$l_2[\![\texttt{stop}]\!] \mid l_1[\![\texttt{goto}\ l_6.\overline{a}\langle d_2\rangle]\!] \mid l_1[\![\texttt{goto}\ l_7.\overline{a}\langle d_2\rangle]\!] \mid$$
$$l_2[\![\tau_{S_2}.(\texttt{goto}\ l_3.\overline{c_1}\langle d_3\rangle + \texttt{goto}\ l_4.\overline{c_2}\langle d_3\rangle)]\!]$$

In this way, the internal computation of step S_2 (τ_{S_2}) is enabled. The data transfer between steps S_1 and S_6 follows the same approach, with the difference that the data is actually transferred to both locations l_6 and l_7 to which step S_6 is mapped.

Remark 1: Taking into account Example 1 and the definition of the step S_6, one could imagine that the natural representation of it should be $S_6 = (a(x) \mid f(y)).\tau_{S_6}$ indicating that the two receiving actions $a(x)$ and $f(y)$ can be executed in parallel in any order, and only when both of them are executed, the internal τ_{S_6} computation can be performed. This kind of modelling, more precise, system $S_6 = (a(x) \mid f(y)).\tau_{S_6}$ is not allowed by the syntax defined in Section II and the one defined in [7], as well. For instance, the term $(a(x) + b(y)).\tau_S$ is also not allowed. Instead, it is modelled as $a(x).\tau_S + b(y).\tau_S$.

Including the mentioned terms into the calculus used in this paper could be done by adding the term $P.P$ into process definition P and in that way allowing any kind of sequencing processes. In that way, one would be able to simplify the representations of the workflows, especially when a step has a big number of input or output actions. On the other side, it would require a complete revision of the rules given in Figures 2 and 3. For instance, the rule (IN) would not be enough because it allows only the execution of the process $a(x).\tau_S$ not of the process $(a(x) \mid f(y)).\tau_S$ where two parallel input actions are gathered to the shared action τ_S.

IV. Discussion

This section contains a discussion of the proposed model and suggests improvements in a few different directions:

1) The description of the workflow, used in this paper, together with its formal representation are quite abstract and with minor modifications allow implementation of the specific features. For instance, Example 1 depicts one of the possible representations of the workflow conditional patterns (choice operator). It could be noticed that when step S_2 make a choice between sending data to the step S_3 or S_4 (location l_3 or l_4), the remaining step is not performing its computation and still waiting for data to arrive. For instance, if data d_3 is sent to the step S_4, step S_3, does not have knowledge about the fact that the decision is made and it is still running waiting for data to process until it is forced to stop or ignored. From the theoretical point of view, one could ignore the fact that the step S_3 is still waiting and state that the workflow computation is completed since the last step of it (in our example, step S_6) is performed. From the practical perspective, this could result in the waste of resources and can impact on the cost of the computation.

In that case, one could think about signaling to the not-chosen step that the decision is made and that it can be terminated. In the calculus defined in Sections II and III, it can be done, for example, by introducing an 'empty' message denoted with a special symbol (for instance $\varnothing$). Like this, the data will be sent to the chosen step, while to the rest of the possible choices, the empty message is outputted. Considering Example 1, the discussed behaviour could be obtained by slightly modifying steps S_2, S_3 and S_4 and by introducing the 'empty' message. The step S_2 could be modelled as (`goto` actions are omitted):

$$S_2 = b(x).\tau_{S_2}.(\overline{c_1}\langle d_3\rangle.\overline{c_2}\langle\varnothing\rangle + \overline{c_2}\langle d_3\rangle.\overline{c_1}\langle\varnothing\rangle)$$

If the chosen path for the data d_3 is transfer through channel c_1, then over the channel c_2 an empty message is sent as a notification that the decision is made. To complete this, step S_3 should be modelled as (`goto` actions are omitted):

$$S_3 = c_1(y).\tau_{S_3}.\overline{e_1}\langle d_4\rangle + c_1(\varnothing).\texttt{stop}$$

The reception of the empty message enables the termination of the step S_3. Introducing the 'empty' message would not imply any major changes in the syntax or semantics of the formal model. The difference would be in the modelling of the choice operator.

Using signaling to notify about the correct or incorrect termination of a process is used already in Erlang[2] in which case signals are not empty messages but containing different values indicating what was the reason for the process termination.

2) The distributed π-calculus [7] does not allow for a direct data transfer if the steps are not on the same location, instead, the additional actions of moving steps to the different locations are executed. While this design decision has a justification when considering the process calculus formalisation, in the workflow setting a different solution may have more impact. For example, by allowing data transfer between different locations, sending data to the step mapped to more locations (i.e. sending data to more locations) could be modelled by data broadcasting. In the calculus presented in this paper, rule (Move) could be substituted with the broadcast rule:

$$\frac{\overline{a}\langle\!\langle d\rangle\!\rangle \xrightarrow{\overline{a}\langle\!\langle d\rangle\!\rangle} stop \quad \forall i\ a(x).P_i \xrightarrow{a(d)} P_i\{d/x\}}{l[\![\overline{a}\langle\!\langle d\rangle\!\rangle]\!] \mid \prod_{i=1}^{n} l_i[\![a(x).P_i]\!] \rightarrow l[\![\texttt{stop}]\!] \mid \prod_{i=1}^{n} l_i[\![P_i\{d/x\}]\!]}$$

where $\overline{a}\langle\!\langle d\rangle\!\rangle$ is a broadcast action and $\prod_{i=1}^{n} l_i[\![a(x).P_i]\!]$ represents all steps to which the data is broadcasted. In this way, the broadcast action is considered atomic, what is not the case in the approach of distributed π-calculus and the rule (Move).

3) The fact that one step can be mapped to different locations imposes the synchronisation between the locations on which the step is executing. It could be executed exactly at the same moment on all locations it is mapped to, or a certain amount of freedom could be left. In the case of distributed π-calculus, there is no synchronisation on the timing of the step execution. It could be implemented, by adding the rule collecting all locations to which step is mapped and forcing them to execute the internal step τ in the same moment.

4) In the majority of workflow systems, the workflow model is entirely independent of the execution topology, meaning that the user has reduced control over the mapping of workflow steps into execution locations. However, if a StreamFlow [14] is considered, it is possible to implement different location topologies. It allows the user to constrain the information flow for security/privacy reasons or to exploit location heterogeneity in the case when the single processing element is expected to be more and more specialised. To be able to model this behaviour, the formal model described in Sections II and III should be equipped with the location topology information. Given a desired topology, it can be represented as a mapping function and added to the syntax as a syntactic sugar of the location. Later the defined function is used as the condition in the semantics rules disabling the application of a certain rule if it is not in the compliance with the mapping function.

5) Loops in the workflow setting could be represented with the recursion operation in the process algebra. To represent loops explicitly and model a setting in which a part of a workflow containing different locations is looping, it is necessary to add a new operator. The recursion is defined as $l[\![P]\!]|_R M_R$ where $l[\![P]\!]$ is the location that triggers the recursion and M_R is the recursion term (in our case, distributed π-calculus system). Then, the necessary rule would be:

$$\frac{P \xrightarrow{\overline{a}\langle d\rangle} \texttt{stop} \quad Q \xrightarrow{a(d)} Q\{d/x\}}{l[\![P]\!]|_R l[\![Q]\!] \rightarrow l[\![Q\{d/x\}]\!]}$$

For instance, consider the workflow in Figure 4. The formal representation would be (`goto` actions are omitted):

[2]https://www.erlang.org/

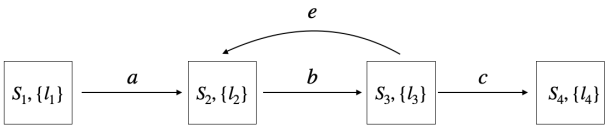

Fig. 4. Example of a workflow with a loop

$$l_1[\![S_1]\!] \mid l_2[\![S_2]\!] \mid l_3[\![S_3]\!]|_R M_R \mid l_4[\![S_4]\!]$$

where

$$S_1 = \tau_{S_1}.\overline{a}\langle d_1\rangle$$
$$S_2 = a(x).\tau_{S_2}.\overline{b}\langle d_2\rangle + e(y).\tau_{S_2}.\overline{b}\langle d_2\rangle$$
$$S_3 = b(x).\tau_{S_3}.(\overline{e}\langle d_3\rangle + \overline{c}\langle d_4\rangle)$$
$$M_R = l_2[\![S_2]\!] \mid l_3[\![S_3]\!]$$
$$S_4 = c(x).\tau_{S_4}$$

From the equations (3) and (4) it can be seen that the recursion term M_R is triggered by the communication on channel e. In this case, the operation $|_R$ guarantees that the recursion terms are executed only when the step S_3 triggers it, not before (by using the classical parallel operator $|$ there would not be such guaranty).

V. Conclusion

This paper explores the possibility of modelling workflow systems with the modified version of the distributed π-calculus [7]. It discusses how adaptable the model could be (thanks to the nature of the π-calculus) and what are the qualifications to be implemented into the basic formalisation.

The fact that mobile systems change their structure by communicating makes distributed π-calculus a good candidate to model the ability to dynamically change workflows on demand. As stated in [17], the formalisation of workflows done with Petri nets does not pose a complete algebra of operations. For instance, there is no concurrency operator that can be used to compose Petri nets to obtain larger Petri nets. On the contrary, compositionality is one of the main features of process algebras and it could be very useful for workflow verification. By increasing the model complexity, the verification of models is more and more difficult. A compositional model would allow for verification of the part of the workflow and then in the next verification step, using the black box for representation of it. In that way, the complexity of the verification tasks would be reduced.

In future work, we shall take into account different proposals mentioned in Section IV and gather them to obtain a formal model able to express a vast range of different workflows. The obtained model could be used to prove different properties of a workflow, like information flow, confluence, correctness, equivalence, etc. Once the formal model is defined, it could be used to model a real case workflow. Additionally, it would simplify the extension of a base model with the new features (for instance fault-tolerance).

References

[1] D. Turi, P. Missier, C. Goble, D. D. Roure and T. Oinn,"Taverna Workflows: Syntax and Semantics," Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007), Bangalore, India, 2007, pp. 441-448, doi: 10.1109/E-SCIENCE.2007.71.

[2] B. Ludscher, I. Altintas, C. Berkley, and el. "Scientific workflow management and the kepler system," Concurrency and Computation: Practice and Experience, Special Issue on Scientific Workflows, 2005.

[3] G. Kahn and D.B. MacQueen, "Coroutines and networks of parallel processes," In IFIP congress, 1977.

[4] C. Ouyang, E. Verbeek, W. M. P. van der Aalst, S. Breutel, M. Dumas, and A. H. M. ter Hofstede, " Formal semantics and analysis of control flow in WS-BPEL," Sci. Comput. Program., 67(2-3):162–198, 2007

[5] W. M. P. van der Aalst, A. H. M. ter Hofstede, "YAWL: yet another workflow language," Information Systems, Volume 30, Issue 4, 2005, Pages 245-275, ISSN 0306-4379

[6] F. Puhlmann, M. Weske, "Using the π-Calculus for Formalizing Workflow Patterns," Business Process Management. BPM 2005, Lecture Notes in Computer Science, vol 3649. Springer, Berlin, Heidelberg.

[7] M. Hennessy, "A Distributed Pi-Calculus. Cambridge: Cambridge University Press," 2005, doi:10.1017/CBO9780511611063

[8] P. Amstutz, M. Mikheev, M. R. Crusoe, N. Tijanić, S. Lampa, et al. , "Existing Workflow systems," Common Workflow Language wiki, GitHub. https://s.apache.org/existing-workflow-systems

[9] W. M. P. van der Aalst,"Three good reasons for using a Petri-net-based workflow management system, pages 161–182. The Kluwer International Series in Engineering and Computer," Kluwer Academic Publishers, Netherlands, 1998.

[10] M. R. Crusoe, S. Abeln, A. Iosup, P. Amstutz, J. Chilton, N. Tijanic, H. Ménager, S. Soiland-Reyes, and C. A. Goble,"Methods included: Standardizing computational reuse and portability with the common workflow language," Communication of the ACM, 2022. doi:10.1145/3486897.

[11] H. Smith, P. Fingar,"Business Process Management – The Third Wave," MeghanKiffer Press, Tampa (2002)

[12] W.M.P. van der Aalst,"Pi calculus versus petri nets: Let us eat "humble pie" rather than further inflate the "pi hype"," (http://is.tm.tue.nl/research/patterns/ download/pi-hype.pdf (2005))

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

[14] I. Colonnelli, B. Cantalupo, I. Merelli, and M. Aldinucci,"StreamFlow: cross-breeding cloud with HPC," IEEE Transactions on Emerging Topics in Computing, 9(4):1723–1737, 2021. doi:10.1109/TETC.2020.3019202.

[15] J. Sroka, J. Hidders, P. Missier, C. Goble,"A formal semantics for the Taverna 2 workflow model," Journal of Computer and System Sciences, Volume 76, Issue 6, 2010, Pages 490-508, ISSN 0022-0000, https://doi.org/10.1016/j.jcss.2009.11.009.

[16] E. Moggi,"Notions of computation and monads," Inf. Comput., 93(1):55–92, 1991.

[17] Y. Dong, Z. Shen-Sheng,"Approach for workflow modeling using π-calculus," Journal of Zhejiang University Science 4 (2003) 643–650

[18] D. Sangiorgi, D. Walker, "The π-calculus: A Theory of Mobile Processes," Paperback edn. Cambridge University Press, Cambridge (2003)

[19] A. Brogi, C. Canal, E. Pimentel, A. Vallecillo, "Formalizing Web Service Choreographies," Proceedings of First International Workshop on Web Services and Formal Methods. Electronic Notes in Theoretical Computer Science, Elsevier (2004)

[20] R. Milner, "Communication and Concurrency," Prentice Hall, 1989

[21] M. P. Forum, "MPI: A Message-Passing Interface Standard", University of Tennessee, USA, Tech. Rep., 1994.

[22] D. Sangiorgi, D. Walker, "The Pi-Calculus - a Theory of Mobile Processes", Cambridge Univ. Press, 2001.