A Foundation for Dynamic Workflow Patterns

Vitus S.W. Lam

***Abstract*—The rationale behind the utilization of workflow patterns is to capture solutions for recurring workflow issues in the domain of business process management. Most of the identified workflow patterns in the literature are static in which their structures are known at design time. This paper takes on the challenge of defining dynamic workflow patterns that their structures rely on runtime factors unknown till the latest possible time. A collection of eight basic and nine derived dynamic workflow patterns is examined from a formal perspective. The π-calculus is adopted as the underlying mathematical foundation for the dynamic workflow patterns. Each derived dynamic workflow pattern is complemented by a real-life example. Our study is of importance in laying the groundwork for reasoning about dynamic workflow patterns.**

***Index Terms*—workflow patterns, π-calculus, control-flow patterns.**

I. Introduction

In today's competitive world, managing business processes in a systematic manner is crucial to the success of an organization. The primary emphasis of business process management (BPM) [1] is on studying the design, analysis, execution, theoretical basis and best practices of business processes. In the BPM field, numerous strategies for constructing business processes are documented as workflow patterns. Seminal papers on workflow patterns encompass [2] and [3].

While most of the previous works on workflow patterns treat only business processes that structures are known at design time, we intend to introduce the notion of dynamic workflow patterns such that their structures can be determined dynamically at runtime. The dynamicity of workflows arises whenever the configuration of a workflow depends directly on the previous activated activities, decision of an activity or external environment at execution time. Unlike ordinary workflow languages and workflow modelling notations such as BPMN [4] that express a bounded number of alternative structures statically, dynamic workflow patterns provide a more intuitive way to represent the alternatives as a dynamically reconfigurable structure.

Apart from the expressiveness issue, this paper is motivated by the need for addressing both the correctness problem and semantic challenge of the dynamic workflow patterns. To facilitate the reasoning about business processes comprising dynamic workflow patterns as well as the formalization of the dynamic workflow patterns, a process-algebraic framework based on the π-calculus [5], [6] is proposed in our prior work [7]. This study advances our previous attempt by augmenting it with new dynamic workflow patterns, concrete examples and correctness proofs.

The rest of the paper is organized as follows. Section 2 reviews previous contributions in the area. Section 3 provides a brief account of the π-calculus. An overview of non-dynamic workflow patterns is given in Section 4. Section 5 offers a detailed discussion of both basic and derived dynamic workflow patterns as well as their π-calculus encodings. The soundness of the π-calculus representations is examined in Section 6. Concluding comments and future work are outlined in Section 7.

V. Lam is with the Information Technology Services, The University of Hong Kong, Hong Kong (e-mail: vitus.lam@ieee.org).

II. Related Work

Dynamic processes [8] (dynamic workflows [9]) are workflow types and workflow instances that change dynamically as a result of the modifications of their environments. Our work differs from [8] and [9] since our main objective is to identify patterns for dynamic workflow instances.

In [3] and [2], van der Aalst et al. and Russell et al. present collections of control-flow patterns. Unlike [3] and [2] that mainly focus on static workflow patterns, our study concentrates on dynamic workflow patterns. In contrast to the approach of [2] that the execution semantics of each workflow pattern is expressed as Coloured Petri-nets, we define each of them in terms of the π-calculus.

In [10], Decker et al. describe an extension to BPEL named BPEL4Chor for modelling service choreographies. The corresponding π-calculus formalization is studied in [11]. As opposed to [10] and [11], our work pertains to orchestrations in lieu of choreographies. Systematic examinations of declarative approaches, which are based on constraints, are provided in [12] and [13]. These studies are markedly different from ours as the nature of our proposed patterns is imperative rather than declarative. Barros et al. [14] generalize the relationships between events, process instances and conversations to a collection of correlation patterns. A crucial difference is that our propounded patterns merely center around process instances in which their structures are determined at runtime.

Yang and Zhang [15] advocate the modelling, verification and equivalence checking of workflows by a rigorous approach based on the π-calculus. Puhlmann and Weske [16] use the π-calculus for encoding the workflow patterns proposed in [3]. Xue et al. [17] examine workflow patterns parallel split, multi-choice, arbitrary cycles, multiple instances without priori runtime knowledge and milestone that have multiple BPMN (Business Process Modelling Notation) [18] representations using the π-calculus. Yang and Zhang [19] adopt the π-calculus as the semantic domain of UML 1.4 activity diagrams. Our previous work [20] formalizes UML 2.0 activity diagrams in the form of the π-calculus. The expressiveness of UML 2.0 activity diagrams for implementing workflow patterns is assessed in [21] and [22]. Nevertheless, there is very limited study on (i) dynamic workflow patterns; and (ii) the specification of dynamic workflow patterns using the π-calculus in the literature.

Other closely related works include [23], [24], [25] and [26] that deal with workflow data patterns, workflow resource patterns, service interaction patterns and time patterns,

respectively. A discussion on the appropriateness of adopting the π-calculus as a mathematical foundation for BPM is given in [27]. In [28], an exploration on the suitability of the π-calculus and Petri-nets for Web Service Composition Language is provided. The application of business process patterns to recover business process models is studied in [29].

This paper builds upon and considerably enhances our earlier work [7] in a number of ways. Firstly, four new basic dynamic workflow patterns *dynamic multi-choice*, *dynamic synchronizing merge*, *dynamic multi-merge* and *dynamic h-out-of-i join* and five new derived dynamic workflow patterns *dynamic multi-choice and synchronizing merge*, *dynamic fork and multi-merge*, *dynamic multi-choice and multi-merge*, *dynamic fork and h-out-of-i join* and *dynamic structured loop* are introduced. Secondly, examples are provided for exemplifying how the derived dynamic workflow patterns are used in real-life setting. Finally, a formal treatment of the correctness of the π-calculus encodings is given.

III. The π-Calculus

This section briefly introduces the essence of the π-calculus and is adapted from our previous works in [30] and [20].

The π-calculus is a mobile process calculus which extends Calculus of Communicating Systems (CCS) [31] through the support of name passing. Contrary to CCS in which the interconnection structures of processes are static, the structure of a system in the π-calculus may change dynamically. Sangiorgi [32] develops the idea further by proposing a higher-order π-calculus in which processes may be passed over channels.

We let $\Sigma^{\pi}_{\mathrm{P}}$ be the set of processes ranged over by P_i, Q_i for $i = 1, \ldots, n$, $\Sigma^{\pi}_{\mathrm{N}}$ be the set of channels (names) ranged over by x_i, y_i for $i = 1, \ldots, n$, $\Sigma^{\pi}_{\mathrm{PI}}$ be the set of process identifiers and $\Sigma^{\pi}_{\mathrm{PDEF}}$ be the set of process definitions. A tuple of channels $x_1, x_2, \ldots, x_n$ is abbreviated to $\vec{x}$. Likewise, we write $\vec{y}$ as an abbreviation for $y_1, y_2, \ldots, y_n$. The syntax of π-calculus process expressions and their corresponding semantics are enumerated as follows:

$x(\vec{y}).P$: is an input prefix which receives channels along channel x and continues as process P with $y_1, y_2, \ldots, y_n$ replaced by the received channels. The input prefix $x().P$ is abbreviated as $x.P$.

$\overline{x}\langle\vec{y}\rangle.P$: is an output prefix which sends channels y_1, y_2, $\ldots$, y_n along channel x and continues as process P. The output prefix $\overline{x}\langle\rangle.P$ is abbreviated as $\overline{x}.P$.

$P|Q$: represents concurrent processes P and Q execute in parallel. $\prod_{i=1}^{n} P_i$ abbreviates $P_1|P_2|\ldots|P_n$.

$P + Q$: represents a non-deterministic choice in which either process P or Q proceeds. $\sum_{i=1}^{n} P_i$ abbreviates $P_1 + P_2 + \ldots + P_n$.

$(\boldsymbol{\nu}\vec{x})P$: is a restriction which creates new channels x_1, x_2, $\ldots$, x_n used for communication in process P.

$\mathbf{0}$: is the null process which cannot perform any actions.

$[x = y]P$: is a matching construct which proceeds as process P if channels x and y are identical; otherwise, behaves like the null process.

$\tau.P$: is an unobservable prefix which performs an internal action τ and continues as process P.

$A(x_1, x_2, \ldots, x_n) \stackrel{\mathrm{def}}{=} P$: denotes a process identifier A which takes n parameters and behaves like process P. Process P may contain occurrences of A.

$!P$: is a replication which behaves as an arbitrary number of concurrent processes P execute in parallel.

The input prefix $x(\vec{y}).P$ and restriction $(\boldsymbol{\nu}\vec{x})P$ bind $\vec{y}$ and $\vec{x}$ in P, respectively. Unlike the input prefix, the channels $\vec{y}$ in the output prefix $\overline{x}\langle\vec{y}\rangle.P$ are free.

In the π-calculus, the operational semantics is defined by reduction. The reduction rule

$$\overline{x}\langle y\rangle.P|x(z).Q \longrightarrow P|Q\{y/z\}$$

specifies that an output prefix $\overline{x}\langle y\rangle.P$ and an input prefix $x(z).Q$ which execute in parallel reduce to concurrent processes P and Q where all free occurrences of z in Q are replaced by y. For further discussion on the other reduction rules, the reader is referred to [33].

IV. Workflow Patterns

Broadly speaking, a design pattern is a general solution to a recurring design problem for a particular domain. A set of 23 design patterns documented by Gamma et al. in [34] are dedicated to the field of object-oriented technology. In the BPM community, the workflow patterns proposed by van der Aalst, ter Hofstede, Kiepuszewski and Barros in [3] are a collection of 20 control-flow patterns that are categorized into six groups: basic control flow patterns, advanced branching and synchronization patterns, structural patterns, multiple instance patterns, state-based patterns and cancellation patterns. Russell, ter Hofstede, van der Aalst and Mulyar [2] extend previous work by incorporating 23 new control-flow patterns into the original set of patterns. The use of an enhanced version of the IBM WebSphere Business Modeller to support the application of control-flow patterns for constructing unstructured workflows is exemplified in [35]. This section offers an introduction to the control-flow patterns that are relevant to this paper. The reader is referred to [3] and [2] for a more detailed treatment.

A fork pattern splits a single flow into multiple concurrent flows which allow activities to execute in parallel. A join pattern synchronizes multiple concurrent flows spawned by a fork pattern as a single flow. An exclusive choice pattern, unlike a fork pattern, determines that only one of the outgoing edges is enabled according to the guard conditions of the outgoing edges. We extend this pattern by allowing the decision to be based on a non-deterministic choice in addition to the guard-condition-based selection mechanism. A simple merge pattern converges multiple incoming edges emanating from an exclusive choice. The outgoing edge of the exclusive choice is enabled for each non-simultaneous activation of the incoming edge. A multi-choice pattern enables one or more outgoing edges in accordance to the guard conditions of the outgoing edges. This pattern, like an exclusive choice pattern, is extended to include the non-deterministic selection mechanism. A synchronizing merge pattern, which is called inclusive merge gateway [4] or OR-join in process definition languages, synchronizes all active incoming edges enabled by a multi-choice pattern as a single outgoing edge. A multi-merge pattern converges multiple incoming edges spawned by either a fork or multi-choice pattern. The outgoing edge of the multi-merge is enabled for each activation of the

incoming edge that may occur simultaneously. A h-out-of-i join pattern enables the outgoing edge when the first h incoming edges are activated. All subsequent $i-h$ activated incoming edges are ignored and the h-out-of-i join construct resets. A discriminator is a special case of the h-out-of-i join pattern in which the value of h is 1. A structured loop pattern is a structure that allows the repeated execution of a sequence of activities provided that a pre-guard-condition or post-guard-condition is satisfied. A deferred choice pattern, unlike an exclusive choice pattern, defers the decision by relying on the environment to activate the first activity of one of the outgoing edges. The other outgoing edges, which are not chosen, are withdrawn. An interleaved parallel routing pattern determines how a set of activities is executed in sequence at runtime. The graphical representations of these patterns, which are expressed as BPMN [18] and UML activity diagrams [36], can be found in [18] and [22], [21], respectively.

V. Dynamic Workflow Patterns

The basic dynamic workflow patterns are workflow constructs where their structures are decided at runtime, whereas the derived dynamic workflow patterns specify how the basic patterns can be combined together. We start by describing the basic dynamic workflow patterns and the rest of the section is then devoted to the derived dynamic workflow patterns.

A. Basic Dynamic Workflow Patterns

In this subsection, we further develop the ideas of [3] and [2] by introducing a number of dynamic workflow patterns. The rationale behind dynamic workflow patterns is the structures of some workflows are only known at runtime rather than design time. A typical example is the number of outgoing edges of a fork construct is required to be determined dynamically by a number of factors including workloads, available resources, etc. before the execution of the construct commences. In what follows, we present eight basic dynamic workflow patterns in which more complex patterns are built upon. The semantics of these basic dynamic workflow patterns is defined when they are utilized for specific combinations in Section V-B.

Pattern 1 (Dynamic Fork)**.** *Suppose $2 \leq i \leq n$. A dynamic fork is a fork in which the number of outgoing edges i is determined dynamically at runtime just before the dynamic fork construct is reached.*

Unlike a fork construct, where the degree of parallelism is decided at design time, a dynamic fork construct provides the flexibility to adjust the degree of parallelism based on a wide variety of conditions. Since the number n is an arbitrary natural number greater than two, no pre-defined upper bound is imposed on the number of concurrent instances. As specified in the one-to-many send pattern [25], the number of parties that received messages is unknown at design time. Despite this is similar to the dynamic fork pattern, a fundamental difference between the two is the former pattern relates to choreographies instead of orchestrations.

Pattern 2 (Dynamic Join)**.** *Suppose $2 \leq i \leq n$. A dynamic join is a join in which the decision of the number of incoming edges i resulting from a preceding dynamic fork construct is made at the latest possible time.*

The number of incoming edges to be merged by a dynamic join construct is determined dynamically according to the number of outgoing edges of the corresponding dynamic fork construct.

Pattern 3 (Dynamic Exclusive Choice)**.** *Suppose $2 \leq i \leq n$. A dynamic exclusive choice is an exclusive choice in which the number of alternatives i is determined at runtime before the execution of the dynamic exclusive choice construct starts.*

Pattern 4 (Dynamic Simple Merge)**.** *Suppose $2 \leq i \leq n$. A dynamic simple merge is a simple merge in which the number of incoming edges i resulting from a preceding dynamic exclusive choice construct is known at some point prior to the firing of the dynamic simple merge construct.*

A dynamic exclusive choice construct is intended to handle the situation that the number of available choices is based on previously executed activities. As an example, the number of alternatives is affected by the amount of available resources as a result of the resources consumed by the executed activities. The number of incoming edges of a dynamic simple merge construct, like a dynamic join construct related to a dynamic fork construct, depends on an associated dynamic exclusive choice construct.

Pattern 5 (Dynamic Multi-choice)**.** *Suppose $2 \leq i \leq n$. A dynamic multi-choice is a multi-choice in which the determination of the number of alternatives i is deferred to the last possible moment.*

Pattern 6 (Dynamic Synchronizing Merge)**.** *Suppose $2 \leq i \leq n$. A dynamic synchronizing merge is a synchronizing merge in which the number of incoming edges i emanating from a preceding dynamic multi-choice is not known until runtime.*

The major difference between dynamic multi-choice and dynamic exclusive choice as well as dynamic synchronizing merge and dynamic simple merge is the former ones allow the enablement of more than one edges.

Pattern 7 (Dynamic Multi-merge)**.** *Suppose $2 \leq i \leq n$. A dynamic multi-merge is a multi-merge in which the number of incoming edges i following either from a dynamic fork or multi-choice construct specified earlier in the model is determined at runtime.*

The subsequent activity of a dynamic multi-merge construct is activated at most certain number of times depending on the number of incoming edges that is dynamically determined.

Pattern 8 (Dynamic h-out-of-i Join)**.** *Suppose $2 \leq i \leq n$. A dynamic h-out-of-i join is a h-out-of-i join in which the decision of the number of incoming edges i preceded by a dynamic fork construct is made before the dynamic h-out-of-i join is enabled.*

A dynamic h-out-of-i join construct eliminates the restriction on the constant number of incoming edges. The number of blocked incoming edges caused by the firing of

the construct increases as the number of incoming edges increases given that the value of h remains unchanged.

Definition 1 (Process Diagram)**.** *A process diagram is a 12-tuple* $PD = (S_{\mathrm{A}}, S_{\mathrm{DF}}, S_{\mathrm{DJ}}, S_{\mathrm{DEC}}, S_{\mathrm{DSM}}, S_{\mathrm{DMC}}, S_{\mathrm{DSynM}}, S_{\mathrm{DMM}}, S_{\mathrm{DHIJ}}, S_{\mathrm{E}}, S_{\mathrm{GC}}, \Phi_{\mathrm{GC}})$ *where*

- S_{A} *is a set of activities;*
- S_{DF} *is a set of dynamic fork constructs defined in Pattern 1;*
- S_{DJ} *is a set of dynamic join constructs defined in Pattern 2;*
- S_{DEC} *is a set of dynamic exclusive choice constructs defined in Pattern 3;*
- S_{DSM} *is a set of dynamic simple merge constructs defined in Pattern 4;*
- S_{DMC} *is a set of dynamic multi-choice constructs defined in Pattern 5;*
- S_{DSynM} *is a set of dynamic synchronizing merge constructs defined in Pattern 6;*
- S_{DMM} *is a set of dynamic multi-merge constructs defined in Pattern 7;*
- S_{DHIJ} *is a set of dynamic h-out-of-i join constructs defined in Pattern 8;*
- $S_{\mathrm{E}} \subseteq (S_{\mathrm{A}} \times S_{\mathrm{A}}) \cup (S_{\mathrm{A}} \times S_{\mathrm{CN}}) \cup (S_{\mathrm{CN}} \times S_{\mathrm{A}}) \cup (S_{\mathrm{CN}} \times S_{\mathrm{CN}})$ *is a set of directed edges such that* $S_{\mathrm{CN}} = S_{\mathrm{DF}} \cup S_{\mathrm{DJ}} \cup S_{\mathrm{DEC}} \cup S_{\mathrm{DSM}} \cup S_{\mathrm{DMC}} \cup S_{\mathrm{DSynM}} \cup S_{\mathrm{DMM}} \cup S_{\mathrm{DHIJ}}$*;*
- S_{GC} *is a set of guard-conditions; and*
- $\Phi_{\mathrm{GC}} : S_{\mathrm{E}} \rightarrow S_{\mathrm{GC}}$ *specifies for an edge its guard-condition.*

For technical convenience, we define a process diagram in terms of the basic dynamic workflow patterns (Patterns 1–8). These patterns are basic constructs which are not decomposable. A directed edge connects (i) two activities; (ii) an activity and a basic dynamic workflow pattern; or (iii) two basic dynamic workflow patterns.

B. Derived Dynamic Workflow Patterns in the π-Calculus

This subsection offers a discussion of nine derived dynamic workflow patterns based on Definition 1. The proposed patterns build upon the integration of the basic dynamic workflow patterns as well as the structural relationships of activities and edges. We begin by examining the suitability of the π-calculus for expressing the dynamic workflow patterns. This is followed by giving the definitions of a number of functions that are used for specifying the derived dynamic workflow patterns. Then we present the derived patterns and their respective π-calculus representations.

In spite of the popularity of Petri nets in the BPM community, Decker et al. [37] point out that Petri nets are unable to encode all the workflow patterns defined in [3]. On the contrary, Puhlmann and Weske [16] successfully transform all these workflow patterns into the π-calculus. Additionally, the formalizations of service invocations, correlations and service interaction patterns using the π-calculus are explored in [38] and [37]. Given these success stories on the adoption of the π-calculus for modelling various sorts of patterns, the utilization of the π-calculus for formalizing the dynamic workflow patterns is justified.

Definition 2. *Suppose* $\mathbb{B} = \{true, false\}$ *is a set of Boolean values,* $SS_{\mathrm{GC}} \subseteq S_{\mathrm{GC}}$ *and* $bval \in \mathbb{B}$*. The function* $extractGC : 2^{S_{\mathrm{GC}}} \times \mathbb{B} \rightarrow 2^{S_{\mathrm{GC}}}$ *defined by* $extractGC(SS_{\mathrm{GC}}, bval) = \{x | x \in SS_{\mathrm{GC}} \wedge$ *the value of* x *equals bval*$\}$ *returns the guard conditions with Boolean value* $bval$*.*

The function $extractGC$ retains all the guard-conditions that values are equal to the Boolean value $bval$ as specified in the parameter.

Definition 3. *Suppose* $\Gamma = \{\mathrm{A}, \mathrm{DF}, \mathrm{DJ}, \mathrm{DEC}, \mathrm{DSM}, \mathrm{DMC}, \mathrm{DSynM}, \mathrm{DMM}, \mathrm{DHIJ}, \mathrm{E}, \mathrm{GC}\}$*. The functions* $\psi_i : S_i \rightarrow \Sigma^{\pi}_{\{\mathrm{PI,N}\}}$ *transform an instance of a graphical element* $\sigma \in S_i$ *of a process diagram into a process identifier or channel of the* π*-calculus where* $i \in \Gamma$ *and* $\Sigma^{\pi}_{\{\mathrm{PI,N}\}}$ *is defined as:*

$$\Sigma^{\pi}_{\{\mathrm{PI,N}\}} = \begin{cases} \Sigma^{\pi}_{\mathrm{PI}} & \text{if } i \in \Gamma \setminus \{\mathrm{E}, \mathrm{GC}\} \\ \Sigma^{\pi}_{\mathrm{N}} & \text{if } i \in \{\mathrm{E}, \mathrm{GC}\}. \end{cases}$$

An activity and a basic dynamic workflow pattern are modelled as a process identifier, whereas an edge and a guard-condition are represented as a channel.

1) Dynamic Fork and Dynamic Join Pattern: The intuition behind the dynamic fork and dynamic join pattern is a dynamic join pattern is a building block that always associates with a dynamic fork pattern. The dynamic fork and dynamic join pattern, which combines the dynamic fork and dynamic join constructs (Patterns 1 and 2), provides the ability for a process diagram to model the concurrent execution of activities.

Pattern 9 (Dynamic Fork and Dynamic Join)**.** *Suppose* P_0, P_1, P_2, Q_1, …, Q_n, R_1, …, $R_n \in S_{\mathrm{A}}$, $DF_1 \in S_{\mathrm{DF}}$, $DJ_1 \in S_{\mathrm{DJ}}$, (P_0, P_1), (P_1, DF_1), (DF_1, Q_k), (R_k, DJ_1), $(DJ_1, P_2) \in S_{\mathrm{E}}$ *and* $2 \leq i \leq n$ *for* $k = 1, \ldots, i$*. The number of concurrent activities* i *is known at runtime when the activity* P_1 *is performed. The dynamic fork* DF_1 *splits the incoming flow from the activity* P_1 *into* i *outgoing flows to the concurrent activities* $Q_1, \ldots, Q_i$ *in which all flows finally connect to the dynamic join* DJ_1*. The dynamic join* DJ_1 *synchronizes all incoming flows from the activities* $R_1, \ldots, R_i$ *as a single outgoing flow to the activity* P_2*.*

The incoming edge of the activity P_1 is represented as (P_0, P_1). Likewise, the incoming edge of the dynamic fork DF_1 and the outgoing edge of the dynamic join DJ_1 are represented by (P_1, DF_1) and (DJ_1, P_2), respectively. The actual connections between concurrent activities $Q_1, \ldots, Q_i$ and concurrent activities $R_1, \ldots, R_i$ are not specified since they vary from one scenario to another. A concrete example, which illustrates how the dynamic fork and dynamic join are linked up by means of a collection of concurrent activities, is given in Example 1.

Definition 4. *Suppose a dynamic fork and dynamic join pattern,* $\psi_{\mathrm{E}}((P_0, P_1)) = p_1$, $\psi_{\mathrm{E}}((P_1, DF_1)) = df_1$, $\psi_{\mathrm{E}}((DF_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k, DJ_1)) = dj_{1,k}$, $\psi_{\mathrm{E}}((DJ_1, P_2)) = p_2$, $eval$ *is a channel for evaluating the number of concurrent activities,* $val_2, \ldots, val_n$ *are channels for representing the possible number of concurrent activities,* $\psi_{\mathrm{A}}(P_1) = P_1(p_1, p_2, eval, val_2, \ldots, val_n, df_1, q_1, \ldots, q_n, dj_{1,1}, \ldots, dj_{1,n})$*,* $\psi_{\mathrm{DF}}(DF_1) = DF_1(df_1, q_1, \ldots, q_i)$ *and* $\psi_{\mathrm{DJ}}(DJ_1) = DJ_1(dj_{1,1}, \ldots, dj_{1,i}, p_2)$*. The encodings of the activity* P_1*, dynamic fork* DF_1 *and dynamic join* DJ_1

are expressed as:

$$P_1(p_1, p_2, eval, val_2, \ldots, val_n, df_1, q_1, \ldots, q_n, dj_{1,1}, \ldots, dj_{1,n}) \stackrel{\text{def}}{=} (\boldsymbol{\nu} rs)p_1.\tau.\overline{eval}\langle rs\rangle.rs(x). \sum_{i=2}^{n}[x = val_i](DF_1(df_1, q_1, \ldots, q_i)| DJ_1(dj_{1,1}, \ldots, dj_{1,i}, p_2)| \overline{df_1}.\mathbf{0})$$

$$DF_1(df_1, q_1, \ldots, q_i) \stackrel{\text{def}}{=} df_1.\prod_{j=1}^{i}\overline{q_j}.\mathbf{0}$$

$$DJ_1(dj_{1,1}, \ldots, dj_{1,i}, p_2) \stackrel{\text{def}}{=} (\boldsymbol{\nu} received)(\prod_{j=1}^{i} dj_{1,j}.\overline{received}.\mathbf{0}|\underbrace{received.\cdots.received}_{i\ times}.\overline{p_2}.\mathbf{0}).$$

The functions ψ_{E}, ψ_{A}, ψ_{DF} and ψ_{DJ} are defined in Definition 3. According to Definition 3, the activity P_1 is modelled as a process identifier $P_1(p_1, p_2,\ eval,\ val_2, \ldots, val_n,\ df_1,\ q_1, \ldots, q_n,\ dj_{1,1}, \ldots, dj_{1,n})$. The receipt of control flow by the process P_1 is modelled as the receipt of a signal along the channel p_1. The execution of the process P_1 is represented by the internal action τ. The determination of the number of outgoing flows of the dynamic fork DF_1 and the number of incoming flows of the dynamic join DJ_1 is denoted in the π-calculus by sending a channel rs along the channel $eval$, receiving a decision along the channel rs and determining the degree of parallelism using matching constructs. The numeric values i are expressed as the channels val_i for $i = 2$ to n. The passing of control flow to the dynamic fork DF_1 is encoded as $\overline{df}.\mathbf{0}$.

The process DF_1 splits a single incoming control flow modelled as df_1 into multiple outgoing control flows represented as $\overline{q_j}$ for $j = 1$ to i. The process DF_1 evolves to the null process only after sending out all the control flows.

The process DJ_1 synchronizes multiple incoming control flows denoted by $dj_{1,j}$ for $j = 1$ to i as a single outgoing control flow represented by $\overline{p_2}$. The channel $received$ prevents the passing of control flow to the subsequent activity before receiving all incoming control flows.

Example 1. *After determining the number of clerks required for handling insurance claims according to the number of claims received and performance pledges, a corresponding number of* ***evaluate claim*** *activities is triggered simultaneously. The completion of all* ***evaluate claim*** *activities enables the activity* ***endorse assessment****.*

Let P_0, $NoOfClerks$, $EvalClaim_1, \ldots, EvalClaim_n$, $EndorseAssess \in S_{\mathrm{A}}$, $DF_1 \in S_{\mathrm{DF}}$, $DJ_1 \in S_{\mathrm{DJ}}$, $(P_0, NoOfClerks)$, $(NoOfClerks, DF_1)$, $(DF_1, EvalClaim_k)$, $(EvalClaim_k, DJ_1)$, $(DJ_1, EndorseAssess) \in S_{\mathrm{E}}$ *and* $2 \leq i \leq n$ *for* $k = 1, \ldots, i$. *We define* $\psi_{\mathrm{E}}((P_0, NoOfClerks)) = noofclerks$, $\psi_{\mathrm{E}}((NoOfClerks, DF_1) = df_1$, $\psi_{\mathrm{E}}((DF_1, EvalClaim_k)) = evalclaim_k$, $\psi_{\mathrm{E}}((EvalClaim_k, DJ_1)) = dj_{1,k}$, $\psi_{\mathrm{E}}((DJ_1, EndorseAssess)) = endorseassess$, $evalclerks$ *as a channel for determining the number of evaluate claim activities,* $val_2, \ldots, val_n$ *as channels for representing the possible number of evaluate claim activities,* $\psi_{\mathrm{A}}(NoOfClerks) = NoOfClerks(noofclerks, endorseassess, evalclerks, val_2, \ldots, val_n, df_1, evalclaim_1, \ldots, evalclaim_n, dj_{1,1}, \ldots, dj_{1,n})$, $\psi_{\mathrm{DF}}(DF_1) = DF_1(df_1, evalclaim_1, \ldots, evalclaim_i)$, $\psi_{\mathrm{DJ}}(DJ_1) = DJ_1(dj_{1,1}, \ldots, dj_{1,i}, endorseassess)$ *and* $\psi_{\mathrm{A}}(EvalClaim_k) = EvalClaim_k(evalclaim_k, dj_{1,k})$. *The activity* $NoOfClerks$, *dynamic fork* DF_1, *dynamic join* DJ_1 *and activities* $EvalClaim_k$ *are modelled in the* π*-calculus as shown below:*

$$NoOfClerks(noofclerks, endorseassess, evalclerks, val_2, \ldots, val_n, df_1, evalclaim_1, \ldots, evalclaim_n, dj_{1,1}, \ldots, dj_{1,n}) \stackrel{\text{def}}{=} (\boldsymbol{\nu} rs)noofclerks.\tau.\overline{evalclerks}\langle rs\rangle.rs(x). \sum_{i=2}^{n}[x = val_i](DF_1(df_1, evalclaim_1, \ldots, evalclaim_i)| DJ_1(dj_{1,1}, \ldots, dj_{1,i}, endorseassess)| \prod_{k=1}^{i} EvalClaim_k(evalclaim_k, dj_{1,k})| \overline{df_1}.\mathbf{0})$$

$$DF_1(df_1, evalclaim_1, \ldots, evalclaim_i) \stackrel{\text{def}}{=} df_1.\prod_{j=1}^{i}\overline{evalclaim_j}.\mathbf{0}$$

$$DJ_1(dj_{1,1}, \ldots, dj_{1,i}, endorseassess) \stackrel{\text{def}}{=} (\boldsymbol{\nu} received)(\prod_{j=1}^{i} dj_{1,j}.\overline{received}.\mathbf{0}|\underbrace{received.\cdots.received}_{i\ times}.\overline{endorseassess}.\mathbf{0})$$

$$EvalClaim_k(evalclaim_k, dj_{1,k}) \stackrel{\text{def}}{=} evalclaim_k.\tau.\overline{dj_{1,k}}.\mathbf{0}.$$

2) Dynamic Exclusive Choice and Dynamic Simple Merge Pattern: The dynamic exclusive choice and dynamic simple merge pattern relates a dynamic exclusive choice construct (Pattern 3) with a dynamic simple merge construct (Pattern 4). Only one control flow is passed from the dynamic exclusive choice construct to the dynamic simple merge construct.

Pattern 10 (Dynamic Exclusive Choice and Dynamic Simple Merge)**.** *Suppose* P_0, P_1, P_2, $Q_1, \ldots, Q_n$, $R_1, \ldots, R_n \in S_A$, $DEC_1 \in S_{\mathrm{DEC}}$, $DSM_1 \in S_{\mathrm{DSM}}$, $gc_1, \ldots, gc_n \in S_{\mathrm{GC}}$, (P_0, P_1), (P_1, DEC_1), (DEC_1, Q_k), (R_k, DSM_1), $(DSM_1, P_2) \in S_{\mathrm{E}}$, $2 \leq i \leq n$ *and* $|\ extractGC(\{gc_1, \ldots, gc_i\}, true)\ | = 1$ *for* $k = 1, \ldots, i$. *The activity* P_1 *determines at runtime the number of alternative activities* i *that is available for selection. The outgoing flows of the alternative activities finally terminate at the dynamic simple merge* DSM_1. *One of the alternative activities* $Q_1, \ldots, Q_i$ *is then selected either non-deterministically by the dynamic exclusive choice* DEC_1 *or deterministically by the dynamic exclusive choice* DEC_1 *based on dynamically-generated guard-conditions* $gc_1, \ldots, gc_i$. *The dynamic simple merge* DSM_1 *activates the activity* P_2 *when the execution of one*

of the alternative activities $R_1, \ldots, R_i$ is completed.

Definition 5. *Suppose a dynamic exclusive choice and dynamic simple merge pattern, $\psi_{\mathrm{E}}((P_0, P_1)) = p_1$, $\psi_{\mathrm{E}}((P_1, DEC_1)) = dec_1$, $\psi_{\mathrm{E}}((DEC_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k, DSM_1)) = dsm_{1,k}$, $\psi_{\mathrm{E}}((DSM_1, P_2)) = p_2$, $eval$ is a channel for determining the number of alternative activities, $val_2, \ldots, val_n$ are channels for modelling the possible number of alternative activities, $\psi_{\mathrm{A}}(P_1) = P_1(p_1, p_2, eval, val_2, \ldots, val_n, dec_1, q_1, \ldots, q_n, dsm_{1,1}, \ldots, dsm_{1,n})$, $\psi_{\mathrm{DEC}}(DEC_1) = DEC_1(dec_1, q_1, \ldots, q_i)$ and $\psi_{\mathrm{DSM}}(DSM_1) = DSM_1(dsm_{1,1}, \ldots, dsm_{1,i}, p_2)$. The activity P_1, dynamic exclusive choice DEC_1 that is non-deterministic and dynamic simple merge DSM_1 are specified in the π-calculus as:*

$$
\begin{aligned}
&P_1(p_1, p_2, eval, val_2, \ldots, val_n, dec_1, q_1, \ldots, q_n,\\
&\quad dsm_{1,1}, \ldots, dsm_{1,n}) \stackrel{\text{def}}{=}\\
&\quad (\boldsymbol{\nu} rs)p_1.\tau.\overline{eval}\langle rs\rangle.rs(x).\\
&\quad \sum_{i=2}^{n}[x = val_i](DEC_1(dec_1, q_1, \ldots, q_i)|\\
&\quad DSM_1(dsm_{1,1}, \ldots, dsm_{1,i}, p_2)|\overline{dec_1}.\mathbf{0})\\
&DEC_1(dec_1, q_1, \ldots, q_i) \stackrel{\text{def}}{=} dec_1.\sum_{j=1}^{i}\overline{q_j}.\mathbf{0}\\
&DSM_1(dsm_{1,1}, \ldots, dsm_{1,i}, p_2) \stackrel{\text{def}}{=}\\
&\quad \sum_{j=1}^{i} dsm_{1,j}.\overline{p_2}.\mathbf{0}.
\end{aligned}
$$

The process DEC_1 models the dynamic exclusive choice by using a non-deterministic choice in which only one of the output prefixes $\overline{q_j}.\mathbf{0}$ for $j = 1$ to i proceeds. The dynamic simple merge is represented as the process DSM_1. The behaviour of merging multiple incoming edges that are not enabled at the same time is encoded in the π-calculus as a non-deterministic choice in which only one of the expressions $dsm_{1,j}.\overline{p_2}.\mathbf{0}$ for $j = 1$ to i executes.

Example 2. *In accordance to the complexity of a presentation topic, a decision on the number of students for a group is made. One of the students is randomly selected and an associated **assign as team leader** activity is initiated. The activity **schedule a team meeting** is enabled when one of the **assign as team leader** activities is completed.*

Since the π-calculus encodings of Example 2 can be obtained directly by applying Definition 5 as illustrated in Example 1, the respective π-calculus representations are omitted here.

Definition 6. *Suppose a dynamic exclusive choice and dynamic simple merge pattern, $\psi_{\mathrm{E}}((P_0, P_1)) = p_1$, $\psi_{\mathrm{E}}((P_1, DEC_1)) = dec_1$, $\psi_{\mathrm{E}}((DEC_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k, DSM_1)) = dsm_{1,k}$, $\psi_{\mathrm{E}}((DSM_1, P_2)) = p_2$, $eval$ is a channel for determining the number of alternative activities, $val_2, \ldots, val_n$ are channels for modelling the possible number of alternative activities, $\Phi_{\mathrm{GC}}((DEC_1, Q_k)) = gc_k$, $\psi_{\mathrm{A}}(P_1) = P_1(p_1, p_2, eval, val_2, \ldots, val_n, dec_1, q_1, \ldots, q_n, dsm_{1,1}, \ldots, dsm_{1,n}, gc_1, \ldots, gc_n, true)$, $\psi_{\mathrm{DEC}}(DEC_1) = DEC_1(dec_1, q_1, \ldots, q_i, gc_1, \ldots, gc_i, true)$, $\psi_{\mathrm{DSM}}(DSM_1) = DSM_1(dsm_{1,1}, \ldots, dsm_{1,i}, p_2)$ and $\psi_{\mathrm{GC}}\ (gc_k) = gc_k$. The activity P_1, dynamic exclusive choice DEC_1 that is deterministic and dynamic simple merge DSM_1 are represented as:*

$$
\begin{aligned}
&P_1(p_1, p_2, eval, val_2, \ldots, val_n, dec_1, q_1, \ldots, q_n,\\
&\quad dsm_{1,1}, \ldots, dsm_{1,n}, gc_1, \ldots, gc_n, true) \stackrel{\text{def}}{=}\\
&\quad (\boldsymbol{\nu} rs)p_1.\tau.\overline{eval}\langle rs\rangle.rs(x).\\
&\quad \sum_{i=2}^{n}[x = val_i](DEC_1(dec_1, q_1, \ldots, q_i, gc_1, \ldots, gc_i,\\
&\quad true)|DSM_1(dsm_{1,1}, \ldots, dsm_{1,i}, p_2)|\overline{dec_1}.\mathbf{0})\\
&DEC_1(dec_1, q_1, \ldots, q_i, gc_1, \ldots, gc_i, true) \stackrel{\text{def}}{=}\\
&\quad dec_1.(\boldsymbol{\nu} x_1, \ldots, x_i, bval_1, \ldots, bval_i)\\
&\quad ((\prod_{j=1}^{i}\overline{gc_j}\langle x_j\rangle.x_j(y_j).\overline{bval_j}\langle y_j\rangle.\mathbf{0})|\\
&\quad bval_1(z_1).\cdots.bval_i(z_i).\\
&\quad \sum_{j=1}^{i}[z_j = true]\overline{q_j}.\mathbf{0})\\
&DSM_1(dsm_{1,1}, \ldots, dsm_{1,i}, p_2) \stackrel{\text{def}}{=}\\
&\quad \sum_{j=1}^{i} dsm_{1,j}.\overline{p_2}.\mathbf{0}.
\end{aligned}
$$

Unlike Definition 5 that the selection of alternative activities is non-deterministic, the choice in Definition 6 is deterministic. The evaluations of the guard-conditions are modelled as the output prefixes $\overline{gc_j}\langle x_j\rangle$ for $j = 1$ to i. The sequence of input actions $bval_1(z_1).\cdots.bval_i(z_i)$ collects all the returned values of the evaluations. The determination of the guard-condition that holds is by means of the matching constructs $[z_j = true]$ for $j = 1$ to i.

Example 3. *The number of alternative flights is determined by the destination for vacation in which one of the itineraries is chosen according to conditions based on price and number of changes required. A corresponding activity **confirm itinerary** is activated and activity **input credit card details** follows immediately when any one of the **confirm itinerary** activities is finished.*

3) Dynamic Multi-choice and Dynamic Synchronizing Merge Pattern: The dynamic multi-choice and dynamic synchronizing merge pattern allows both the number of alternative activities and the number of activated outgoing edges to be determined at runtime. It builds on Patterns 5 and 6.

Pattern 11 (Dynamic Multi-choice and Dynamic Synchronizing Merge)**.** *Suppose $P_0, P_1, P_2, Q_1, \ldots, Q_n, R_1, \ldots, R_n \in S_{\mathrm{A}}$, $DMC_1 \in S_{\mathrm{DMC}}$, $DSynM_1 \in S_{\mathrm{DSynM}}$, $gc_1, \ldots, gc_n \in S_{\mathrm{GC}}$, (P_0, P_1), (P_1, DMC_1), (DMC_1, Q_k), $(R_k, DSynM_1)$, $(DSynM_1, P_2) \in S_{\mathrm{E}}$, $2 \leq i \leq n$ and $|\, extractGC(\{gc_1, \ldots, gc_i\}, true)\,| \geq 1$ for $k = 1, \ldots, i$. The activity P_1 evaluates the number of alternative activities i that is available for selection at runtime. The outgoing flows of the alternative activities finally terminate at the dynamic synchronizing merge $DSynM_1$. One or more of the alternative activities $Q_1, \ldots, Q_i$ are then selected either non-deterministically by the dynamic multi-choice DMC_1 or*

deterministically by the dynamic multi-choice DMC_1 based on dynamically-generated guard-conditions $gc_1, \ldots, gc_i$. The dynamic synchronizing merge $DSynM_1$ synchronizes the incoming flows from the activities $R_1, \ldots, R_i$ that are activated as a single outgoing flow to the activity P_2.

One of the prominent challenges in the BPM community is to devise a behavioural semantics with a linear-time complexity for synchronizing merge. Dumas et al. [39] and Völzer [40] propound, respectively, algorithms for determining the activation of a synchronizing merge in linear time. Unlike these prior efforts, we advocate the use of two channels $activate$ and $nactivate$ for signifying whether an incoming edge of a synchronizing merge is activated or not in Definition 7.

Definition 7. *Suppose a dynamic multi-choice and dynamic synchronizing merge pattern, $\psi_{\mathrm{E}}((P_0, P_1)) = p_1$, $\psi_{\mathrm{E}}((P_1, DMC_1)) = dmc_1$, $\psi_{\mathrm{E}}((DMC_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k, DSynM_1)) = dsynm_{1,k}$, $\psi_{\mathrm{E}}((DSynM_1, P_2)) = p_2$, $eval$ is a channel for determining the number of alternative activities, $val_2, \ldots, val_n$ are channels for modelling the possible number of alternative activities, $notReachLim_{i-1}$ is a channel for testing whether the limit is reached in which a maximum number of $i-1$ outgoing edges may not be activated, $count_i$ is a channel for counting the number of outgoing edges in which there is already a decision on whether the outgoing edge is activated or not, $activate_i$ is a channel for signifying that the ith outgoing edge is activated, $nactivate_i$ is a channel for representing that the ith outgoing edge is not activated, $killNRLim_{i-1}$ is a channel for terminating the process $NotReachLim_{i-1}$, $\psi_{\mathrm{A}}(P_1) = P_1(p_1, p_2, eval, val_2, \ldots, val_n, dmc_1, q_1, \ldots, q_n, notReachLim_{i-1}, count_i, killNRLim_{i-1}, dsynm_{1,1}, \ldots, dsynm_{1,n}, activate_1, \ldots, activate_n, nactivate_1, \ldots, nactivate_n)$, $\psi_{\mathrm{DMC}}(DMC_1) = DMC_1(dmc_1, q_1, \ldots, q_i, notReachLim_{i-1}, count_i, activate_1, \ldots, activate_i, nactivate_1, \ldots, nactivate_i)$ and $\psi_{\mathrm{DSynM}}(DSynM_1) = DSynM_1(dsynm_{1,1}, \ldots, dsynm_{1,i}, p_2, activate_1, \ldots, activate_i, nactivate_1, \ldots, nactivate_i)$. The representations of the activity P_1, dynamic multi-choice DMC_1 that is non-deterministic and dynamic synchronizing merge $DSynM_1$ are given by:*

$$
\begin{aligned}
&P_1(p_1, p_2, eval, val_2, \ldots, val_n, dmc_1, q_1, \ldots, q_n,\\
&\quad notReachLim_{i-1}, count_i, killNRLim_{i-1},\\
&\quad dsynm_{1,1}, \ldots, dsynm_{1,n}, activate_1, \ldots, activate_n,\\
&\quad nactivate_1, \ldots, nactivate_n) \stackrel{\text{def}}{=}\\
&\quad (\boldsymbol{\nu} rs) p_1.\tau.\overline{eval}\langle rs\rangle.rs(x).\\
&\quad \sum_{i=2}^{n}[x = val_i](DMC_1(dmc_1, q_1, \ldots, q_i,\\
&\qquad notReachLim_{i-1}, count_i, activate_1, \ldots, activate_i,\\
&\qquad nactivate_1, \ldots, nactivate_i)|\\
&\qquad NotReachLim_{i-1}(notReachLim_{i-1},\\
&\qquad killNRLim_{i-1})|\\
&\qquad Count_i(count_i, killNRLim_{i-1})|\\
&\qquad DSynM_1(dsynm_{1,1}, \ldots, dsynm_{1,i}, p_2,\\
&\qquad activate_1, \ldots, activate_i,\\
&\qquad nactivate_1, \ldots, nactivate_i)|\\
&\qquad \overline{dmc_1}.\mathbf{0})\\
&DMC_1(dmc_1, q_1, \ldots, q_i, notReachLim_{i-1}, count_i,\\
&\quad activate_1, \ldots, activate_i,\\
&\quad nactivate_1, \ldots, nactivate_i) \stackrel{\text{def}}{=}\\
&\quad dmc_1.\prod_{j=1}^{i}(\overline{activate_j}.\overline{q_j}.\overline{count_i}.\mathbf{0} +\\
&\quad \overline{notReachLim_{i-1}}.\overline{nactivate_j}.\overline{count_i}.\mathbf{0})\\
&NotReachLim_{i-1}(notReachLim_{i-1}, killNRLim_{i-1}) \stackrel{\text{def}}{=}\\
&\quad \underbrace{notReachLim_{i-1}.(notReachLim_{i-1}.}\\
&\qquad \underbrace{(\cdots.(notReachLim_{i-1}.}_{i\text{-1 times}}\\
&\quad killNRLim_{i-1}.\mathbf{0} +\\
&\quad \underbrace{killNRLim_{i-1}.\mathbf{0})\cdots) + killNRLim_{i-1}.\mathbf{0})}+\\
&\qquad \underbrace{killNRLim_{i-1}.\mathbf{0}}_{i\text{-1 times}}\\
&Count_i(count_i, killNRLim_{i-1}) \stackrel{\text{def}}{=}\\
&\quad \underbrace{count_i.\cdots.count_i}_{i\ times}.\overline{killNRLim_{i-1}}.\mathbf{0}\\
&DSynM_1(dsynm_{1,1}, \ldots, dsynm_{1,i}, p_2,\\
&\quad activate_1, \ldots, activate_i,\\
&\quad nactivate_1, \ldots, nactivate_i) \stackrel{\text{def}}{=}\\
&\quad (\boldsymbol{\nu} received)\\
&\quad (\prod_{j=1}^{i}(activate_j.dsynm_{1,j}.\overline{received}.\mathbf{0} +\\
&\qquad nactivate_j.\overline{received}.\mathbf{0})|\\
&\qquad \underbrace{received.\cdots.received}_{i\ times}.\overline{p_2}.\mathbf{0}).
\end{aligned}
$$

The dynamic multi-choice and dynamic synchronizing merge pattern requires that one or more activities are selected from i alternative activities. This implies there is a limit that at most $i-1$ outgoing edges are not activated. The process $NotReachLim_{i-1}$ provides a way for testing whether the limit $i-1$ is reached. The process $Count_i$ is a counter for keeping track of the number of outgoing edges that there is a decision on whether the outgoing edge is activated or not. A signal is sent along the channel $killNRLim_{i-1}$ after making all the decisions. The process $NotReachLim_{i-1}$ then evolves to the null process on receipt of a signal along the channel $killNRLim_{i-1}$.

The process DMC_1 inputs a signal along the channel dmc_1 and evolves to i concurrent processes. Each concurrent process then either

(i) sends out an activated signal along the channel $activate_j$, passes the control flow to the subsequent activity and updates the counter; or

(ii) tests whether the limit is reached, sends out a non-activated signal along the channel $nactivate_j$, and updates the counter.

If the activity edge is activated, the process $DSynM_1$ sends out a signal along the channel $received$ upon receiving the control flow. If the activity edge is non-activated, there is

no need to wait for the receipt of the control flow and the process $DSynM_1$ simply sends out a signal along the channel $received$. The control flow is then passed to the activity P_2 on receiving the ith signal along the channel $received$.

Example 4. *The set of documents which is required for supporting an insurance claim is depended on the type of claims. Two activities* ***submit the receipt*** *and* ***submit the photograph of a damaged good*** *are activated out of all possible submissions before the activity* ***assess claim*** *commences.*

Definition 8. *Suppose a dynamic multi-choice and dynamic synchronizing merge pattern,* $\psi_{\rm E}((P_0,\ P_1)) = p_1$, $\psi_{\rm E}((P_1,\ DMC_1)) = dmc_1$, $\psi_{\rm E}((DMC_1,\ Q_k)) = q_k$, $\psi_{\rm E}((R_k,\ DSynM_1)) = dsynm_{1,k}$, $\psi_{\rm E}((DSynM_1,\ P_2)) = p_2$, $eval$ *is a channel for determining the number of alternative activities,* $val_2, \ldots, val_n$ *are channels for modelling the possible number of alternative activities,* $activate_i$ *is a channel for signifying that the* i*th outgoing edge is activated,* $nactivate_i$ *is a channel for representing that the* i*th outgoing edge is not activated,* $\Phi_{\rm GC}((DMC_1, Q_k)) = gc_k$, $\psi_{\rm A}(P_1) = P_1(p_1,\ p_2,\ eval,\ val_2,\ \ldots,\ val_n,\ dmc_1,\ q_1,\ \ldots,\ q_n,\ dsynm_{1,1},\ \ldots,\ dsynm_{1,n},\ activate_1, \ldots, activate_n, nactivate_1, \ldots, nactivate_n,\ gc_1,\ \ldots,\ gc_n,\ true,\ false)$, $\psi_{\rm DMC}(DMC_1) = DMC_1(dmc_1,\ q_1,\ \ldots,\ q_i,\ activate_1,\ \ldots,\ activate_i,\ nactivate_1,\ \ldots,\ nactivate_i,\ gc_1,\ \ldots,\ gc_i,\ true,\ false)$, $\psi_{\rm DSynM}(DSynM_1) = DSynM_1(dsynm_{1,1},\ \ldots,\ dsynm_{1,i},\ p_2,\ activate_1,\ \ldots,\ activate_i,\ nactivate_1,\ \ldots,\ nactivate_i)$ *and* $\psi_{\rm GC}(gc_k) = gc_k$. *The activity* P_1*, dynamic multi-choice* DMC_1 *that is deterministic and dynamic synchronizing merge* $DSynM_1$ *are encoded in the* π*-calculus as:*

$$
\begin{aligned}
&P_1(p_1, p_2, eval, val_2, \ldots, val_n, dmc_1, q_1, \ldots, q_n,\\
&\quad dsynm_{1,1}, \ldots, dsynm_{1,n}, activate_1, \ldots, activate_n,\\
&\quad nactivate_1, \ldots, nactivate_n, gc_1, \ldots, gc_n,\\
&\quad true, false) \stackrel{\rm def}{=}\\
&\quad (\boldsymbol{\nu} rs) p_1.\tau.\overline{eval}\langle rs\rangle.rs(x).\\
&\quad \sum_{i=2}^{n}[x = val_i](DMC_1(dmc_1, q_1, \ldots, q_i,\\
&\qquad activate_1, \ldots, activate_i,\\
&\qquad nactivate_1, \ldots, nactivate_i,\\
&\qquad gc_1, \ldots, gc_i, true, false)|\\
&\qquad DSynM_1(dsynm_{1,1}, \ldots, dsynm_{1,i}, p_2,\\
&\qquad activate_1, \ldots, activate_i,\\
&\qquad nactivate_1, \ldots, nactivate_i)|\\
&\qquad \overline{dmc_1}.\mathbf{0})\\
&DMC_1(dmc_1, q_1, \ldots, q_i, activate_1, \ldots, activate_i,\\
&\quad nactivate_1, \ldots, nactivate_i, gc_1, \ldots, gc_i,\\
&\quad true, false) \stackrel{\rm def}{=}\\
&\quad dmc_1.(\boldsymbol{\nu} x_1, \ldots, x_i)\\
&\quad (\prod_{j=1}^{i}\overline{gc_j}\langle x_j\rangle.x_j(y_j).\\
&\qquad ([y_j = true]\overline{activate_j}.\overline{q_j}.\mathbf{0} +\\
&\qquad [y_j = false]\overline{nactivate_j}.\mathbf{0}))
\end{aligned}
$$

$$
\begin{aligned}
&DSynM_1(dsynm_{1,1}, \ldots, dsynm_{1,i}, p_2, activate_1,\\
&\quad \ldots, activate_i, nactivate_1, \ldots, nactivate_i) \stackrel{\rm def}{=}\\
&\quad (\boldsymbol{\nu} received)\\
&\quad (\prod_{j=1}^{i}(activate_j.dsynm_{1,j}.\overline{received}.\mathbf{0} +\\
&\qquad nactivate_j.\overline{received}.\mathbf{0})|\\
&\qquad \underbrace{received.\cdots.received}_{i\ times}.\overline{p_2}.\mathbf{0}).
\end{aligned}
$$

The selection of alternative outgoing edges in Definition 7 is non-deterministic, whereas the one in Definition 8 is deterministic. The output prefixes $\overline{gc_j}\langle x_j\rangle$ for $j = 1$ to i model the evaluations of the guard-conditions. The matching constructs $[y_j = true]$ and $[y_j = false]$ for $j = 1$ to i determine which guard-conditions hold.

Example 5. *A variant of Example 4 by adding guard-conditions* ***availability of receipt****,* ***availability of damaged good****, etc.*

4) Dynamic Fork and Dynamic Multi-merge Pattern: The dynamic fork and dynamic multi-merge pattern associates a dynamic fork construct (Pattern 1) with a dynamic multi-merge construct (Pattern 7). Each outgoing control flow of the dynamic fork construct activates the subsequent activity of the dynamic merge construct separately.

Pattern 12 (Dynamic Fork and Dynamic Multi-merge)**.** *Suppose* $P_0,\ P_1,\ P_2,\ Q_1,\ \ldots,\ Q_n,\ R_1,\ \ldots,\ R_n \in S_{\rm A}$, $DF_1 \in S_{\rm DF}$, $DMM_1 \in S_{\rm DMM}$, $(P_0,\ P_1)$, $(P_1,\ DF_1)$, $(DF_1,\ Q_k)$, $(R_k,\ DMM_1)$, $(DMM_1,\ P_2) \in S_{\rm E}$ *and* $2 \leq i \leq n$ *for* $k = 1,\ \ldots,\ i$*. The number of concurrent activities* i *is determined at runtime when the activity* P_1 *is performed. The dynamic fork* DF_1 *splits the incoming flow from the activity* P_1 *into* i *outgoing flows to the concurrent activities* $Q_1,\ \ldots,\ Q_i$ *in which all flows finally connect to the dynamic multi-merge* DMM_1*. The dynamic multi-merge* DMM_1 *activates the activity* P_2 *for each of the completion of the activity* R_k*.*

Definition 9. *Suppose a dynamic fork and dynamic multi-merge pattern,* $\psi_{\rm E}((P_0,\ P_1)) = p_1$, $\psi_{\rm E}((P_1,\ DF_1)) = df_1$, $\psi_{\rm E}((DF_1,\ Q_k)) = q_k$, $\psi_{\rm E}((R_k,\ DMM_1)) = dmm_{1,k}$, $\psi_{\rm E}((DMM_1,\ P_2)) = p_2$, $eval$ *is a channel for determining the number of concurrent activities,* $val_2, \ldots, val_n$ *are channels for modelling the possible number of concurrent activities,* $\psi_{\rm A}(P_1) = P_1(p_1, p_2,\ eval,\ val_2, \ldots, val_n,\ df_1,\ q_1, \ldots, q_n,\ dmm_{1,1}, \ldots, dmm_{1,n})$, $\psi_{\rm DF}(DF_1) = DF_1(df_1,\ q_1,\ \ldots,\ q_i)$ *and* $\psi_{\rm DMM}(DMM_1) = DMM_1(dmm_{1,1},\ \ldots,\ dmm_{1,i},\ p_2)$*. The* π*-calculus specifications of the activity* P_1*, dynamic fork* DF_1 *and dynamic multi-merge* DMM_1 *are given by:*

$$
\begin{aligned}
&P_1(p_1, p_2, eval, val_2, \ldots, val_n, df_1, q_1, \ldots, q_n,\\
&\quad dmm_{1,1}, \ldots, dmm_{1,n}) \stackrel{\rm def}{=}\\
&\quad (\boldsymbol{\nu} rs) p_1.\tau.\overline{eval}\langle rs\rangle.rs(x).\\
&\quad \sum_{i=2}^{n}[x = val_i](DF_1(df_1, q_1, \ldots, q_i)|\\
&\qquad DMM_1(dmm_{1,1}, \ldots, dmm_{1,i}, p_2)|
\end{aligned}
$$

$\overline{df_1}.\mathbf{0})$

$DF_1(df_1, q_1, \ldots, q_i) \stackrel{\text{def}}{=}$

$df_1.\prod_{j=1}^{i}\overline{q_j}.\mathbf{0}$

$DMM_1(dmm_{1,1}, \ldots, dmm_{1,i}, p_2) \stackrel{\text{def}}{=}$

$\prod_{j=1}^{i} dmm_{1,j}.\overline{p_2}.\mathbf{0}.$

The dynamic fork construct is specified by the process DF_1. The outgoing control flows are represented as a parallel composition which consists of the output actions $\overline{q_j}$ for $j = 1$ to i.

The behaviour of the dynamic merge construct is defined by the process DMM_1. Upon receipt of a signal along the channel $dmm_{1,j}$, a signal is sent along the channel p_2.

Example 6. *The number of activity **pack goods** to be executed concurrently is derived according to the number of orders received. The activity **ship goods** is enabled whenever a **pack goods** activity is completed.*

5) Dynamic Multi-choice and Dynamic Multi-merge Pattern: The dynamic multi-choice and dynamic multi-merge pattern combines a dynamic multi-choice construct (Pattern 5) with a dynamic multi-merge construct (Pattern 7). Some outgoing activity edges of the dynamic multi-choice construct are enabled and they trigger the subsequent activity of the dynamic multi-merge construct separately.

Pattern 13 (Dynamic Multi-choice and Dynamic Multi--merge)**.** *Suppose* P_0, P_1, P_2, Q_1, $\ldots$, Q_n, R_1, $\ldots$, $R_n \in S_{\mathrm{A}}$, $DMC_1 \in S_{\mathrm{DMC}}$, gc_1, $\ldots$, $gc_n \in S_{\mathrm{GC}}$, $DMM_1 \in S_{\mathrm{DMM}}$, (P_0, P_1), (P_1, DMC_1), (DMC_1, Q_k), (R_k, DMM_1), $(DMM_1, P_2) \in S_{\mathrm{E}}$, $2 \leq i \leq n$ *and* $|\ extractGC(\{gc_1, \ldots, gc_i\}, true)\ | \geq 1$ *for* $k = 1, \ldots, i$. *The activity* P_1 *evaluates the number of alternative activities* i *that is available for selection at runtime. The outgoing flows of the alternative activities finally terminate at the dynamic multi-merge* DMM_1. *One or more of the alternative activities* $Q_1, \ldots, Q_i$ *are then selected either non-deterministically by the dynamic multi-choice* DMC_1 *or deterministically by the dynamic multi-choice* DMC_1 *based on dynamically-generated guard-conditions* $gc_1, \ldots, gc_i$. *The dynamic multi-merge* DMM_1 *activates the activity* P_2 *for each of the completion of the activity* R_k.

Definition 10. *Suppose a dynamic multi-choice and dynamic multi-merge pattern,* $\psi_{\mathrm{E}}((P_0, P_1)) = p_1$, $\psi_{\mathrm{E}}((P_1, DMC_1)) = dmc_1$, $\psi_{\mathrm{E}}((DMC_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k, DMM_1)) = dmm_{1,k}$, $\psi_{\mathrm{E}}((DMM_1, P_2)) = p_2$, $eval$ *is a channel for determining the number of alternative activities,* $val_2, \ldots, val_n$ *are channels for modelling the possible number of alternative activities,* $notReachLim_{i-1}$ *is a channel for testing whether the limit is reached in which a maximum number of* $i-1$ *outgoing edges may not be activated,* $count_i$ *is a channel for counting the number of outgoing edges in which there is already a decision on whether the outgoing edge is activated or not,* $activate_i$ *is a channel for signifying that the* i*th outgoing edge is activated,* $nactivate_i$ *is a channel for representing that the* i*th outgoing edge is not activated,* $killNRLim_{i-1}$ *is a channel for terminating the process* $NotReachLim_{i-1}$, $\psi_{\mathrm{A}}(P_1) = P_1(p_1, p_2, eval, val_2, \ldots, val_n, dmc_1, q_1, \ldots, q_n, notReachLim_{i-1}, count_i, killNRLim_{i-1}, dmm_{1,1}, \ldots, dmm_{1,n}, activate_1, \ldots, activate_n, nactivate_1, \ldots, nactivate_n)$, $\psi_{\mathrm{DMC}}(DMC_1) = DMC_1(dmc_1, q_1, \ldots, q_i, notReachLim_{i-1}, count_i, activate_1, \ldots, activate_i, nactivate_1, \ldots, nactivate_i)$ *and* $\psi_{\mathrm{DMM}}(DMM_1) = DMM_1(dmm_{1,1}, \ldots, dmm_{1,i}, p_2, activate_1, \ldots, activate_i, nactivate_1, \ldots, nactivate_i)$. *The activity* P_1, *dynamic multi-choice* DMC_1 *that is non-deterministic and dynamic multi-merge* DMM_1 *are defined in the* π*-calculus by:*

$P_1(p_1, p_2, eval, val_2, \ldots, val_n, dmc_1, q_1, \ldots, q_n,$
$notReachLim_{i-1}, count_i, killNRLim_{i-1},$
$dsynm_{1,1}, \ldots, dsynm_{1,n},$
$activate_1, \ldots, activate_n,$
$nactivate_1, \ldots, nactivate_n) \stackrel{\text{def}}{=}$
$(\boldsymbol{\nu} rs)p_1.\tau.\overline{eval}\langle rs\rangle.rs(x).$
$\sum_{i=2}^{n}[x = val_i](DMC_1(dmc_1, q_1, \ldots, q_i,$
$notReachLim_{i-1}, count_i,$
$activate_1, \ldots, activate_i,$
$nactivate_1, \ldots, nactivate_i)|$
$NotReachLim_{i-1}(notReachLim_{i-1},$
$killNRLim_{i-1})|$
$Count_i(count_i, killNRLim_{i-1})|$
$DMM_1(dmm_{1,1}, \ldots, dmm_{1,i}, p_2,$
$activate_1, \ldots, activate_i,$
$nactivate_1, \ldots, nactivate_i)|$
$\overline{dmc_1}.\mathbf{0})$

$DMC_1(dmc_1, q_1, \ldots, q_i, notReachLim_{i-1}, count_i,$
$activate_1, \ldots, activate_i,$
$nactivate_1, \ldots, nactivate_i) \stackrel{\text{def}}{=}$
$dmc_1.\prod_{j=1}^{i}(\overline{activate_j}.\overline{q_j}.\overline{count_i}.\mathbf{0} +$
$\overline{notReachLim_{i-1}}.\overline{nactivate_j}.\overline{count_i}.\mathbf{0})$

$NotReachLim_{i-1}(notReachLim_{i-1}, killNRLim_{i-1}) \stackrel{\text{def}}{=}$
$\underbrace{notReachLim_{i-1}.(notReachLim_{i-1}.}$
$\underbrace{(\cdots.(notReachLim_{i-1}.}_{i\text{-1 times}}$
$killNRLim_{i-1}.\mathbf{0} +$
$\underbrace{killNRLim_{i-1}.\mathbf{0})\cdots)+}$
$\underbrace{killNRLim_{i-1}.\mathbf{0}) + killNRLim_{i-1}.\mathbf{0}}_{i\text{-1 times}}$

$Count_i(count_i, killNRLim_{i-1}) \stackrel{\text{def}}{=}$
$\underbrace{count_i.\cdots.count_i}_{i\ times}.\overline{killNRLim_{i-1}}.\mathbf{0}$

$DMM_1(dmm_{1,1}, \ldots, dmm_{1,i}, p_2,$
$activate_1, \ldots, activate_i,$
$nactivate_1, \ldots, nactivate_i) \stackrel{\text{def}}{=}$

$$\prod_{j=1}^{i}(activate_j.dmm_{1,j}.\overline{p_2}.\mathbf{0} + nactivate_j.\mathbf{0}).$$

The representation of the dynamic multi-choice and dynamic multi-merge pattern based on non-deterministic choice is similar to that of the dynamic multi-choice and dynamic synchronizing merge pattern defined in Definition 7. The only difference is that the output action $\overline{p_2}$ is performed after the input actions $dmm_{1,j}$ for $j = 1$ to i in lieu of the sequence of input actions $\underbrace{received.\cdots.received}_{i\ times}$.

Example 7. *The number of products available for inspection is based on the production line. Some of them are randomly selected for examination simultaneously and the activity* ***mark product as defect*** *is executed whenever a defected product is found.*

Definition 11. *Suppose a dynamic multi-choice and dynamic multi-merge pattern,* $\psi_{\mathrm{E}}((P_0,\ P_1)) = p_1$, $\psi_{\mathrm{E}}((P_1,\ DMC_1)) = dmc_1$, $\psi_{\mathrm{E}}((DMC_1,\ Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k,\ DMM_1)) = dmm_{1,k}$, $\psi_{\mathrm{E}}((DMM_1,\ P_2)) = p_2$, $eval$ *is a channel for determining the number of alternative activities,* $val_2, \ldots, val_n$ *are channels for modelling the possible number of alternative activities,* $activate_i$ *is a channel for signifying that the* i*th outgoing edge is activated,* $nactivate_i$ *is a channel for representing that the* i*th outgoing edge is not activated,* $\Phi_{\mathrm{GC}}((DMC_1, Q_k))=g_k$, $\psi_{\mathrm{A}}(P_1) = P_1(p_1, p_2,\ eval,\ val_2,\ \ldots,\ val_n,\ dmc_1,\ q_1,\ \ldots,\ q_n,\ dmm_{1,1},\ \ldots,\ dmm_{1,n},\ activate_1,\ \ldots,\ activate_n,\ nactivate_1,\ \ldots,\ nactivate_n,\ gc_1,\ \ldots,\ gc_n,\ true,\ false)$, $\psi_{\mathrm{DMC}}(DMC_1) = DMC_1(dmc_1, q_1, \ldots, q_i, activate_1, \ldots, activate_i,\ nactivate_1,\ \ldots,\ nactivate_i,\ gc_1, \ldots, gc_i,\ true,\ false)$, $\psi_{\mathrm{DMM}}(DMM_1) = DMM_1(dmm_{1,1},\ \ldots,\ dmm_{1,i},\ p_2,\ activate_1,\ \ldots,\ activate_i,\ nactivate_1,\ \ldots,\ nactivate_i)$ *and* $\psi_{\mathrm{GC}}(gc_k) = gc_k$. *The activity* P_1*, dynamic multi-choice* DMC_1 *that is deterministic and dynamic multi-merge* DMM_1 *are specified in the* π*-calculus as:*

$$\begin{aligned}
&P_1(p_1, p_2, eval, val_2, \ldots, val_n, dmc_1, q_1, \ldots, q_n,\\
&\quad dmm_{1,1}, \ldots, dmm_{1,n}, activate_1, \ldots, activate_n,\\
&\quad nactivate_1, \ldots, nactivate_n, gc_1, \ldots, gc_n,\\
&\quad true, false) \stackrel{\mathrm{def}}{=}\\
&\quad (\boldsymbol{\nu} rs)p_1.\tau.\overline{eval}\langle rs\rangle.rs(x).\\
&\quad \sum_{i=2}^{n}[x = val_i](DMC_1(dmc_1, q_1, \ldots, q_i,\\
&\qquad activate_1, \ldots, activate_i,\\
&\qquad nactivate_1, \ldots, nactivate_i, gc_1, \ldots, gc_i, true, false)|\\
&\qquad DMM_1(dmm_{1,1}, \ldots, dmm_{1,i}, p_2,\\
&\qquad activate_1, \ldots, activate_i,\\
&\qquad nactivate_1, \ldots, nactivate_i)|\\
&\qquad \overline{dmc_1}.\mathbf{0})\\
&DMC_1(dmc_1, q_1, \ldots, q_i, activate_1, \ldots, activate_i,\\
&\quad nactivate_1, \ldots, nactivate_i, gc_1, \ldots, gc_i,\\
&\quad true, false) \stackrel{\mathrm{def}}{=}\\
&\quad dmc_1.(\boldsymbol{\nu} x_1, \ldots, x_i)\\
&\qquad (\prod_{j=1}^{i}\overline{gc_j}\langle x_j\rangle.x_j(y_j).\\
&\qquad\quad ([y_j = true]\overline{activate_j}.\overline{q_j}.\mathbf{0} +\\
&\qquad\quad [y_j = false]\overline{nactivate_j}.\mathbf{0}))\\
&\quad DMM_1(dmm_{1,1}, \ldots, dmm_{1,i}, p_2,\\
&\qquad activate_1, \ldots, activate_i,\\
&\qquad nactivate_1, \ldots, nactivate_i) \stackrel{\mathrm{def}}{=}\\
&\qquad \prod_{j=1}^{i}(activate_j.dmm_{1,j}.\overline{p_2}.\mathbf{0} + nactivate_j.\mathbf{0}).
\end{aligned}$$

Likewise, the encodings of the dynamic multi-choice and dynamic multi-merge pattern based on deterministic choice and the dynamic multi-choice and dynamic synchronizing merge pattern defined in Definition 8 are similar. The discrepancy lies in the output action $\overline{p_2}$ is executed after the input actions $dmm_{1,j}$ for $j = 1$ to i rather than the sequence of input actions $\underbrace{received.\cdots.received}_{i\ times}$.

Example 8. *A variation of Example 7 by selecting all those products with serial numbers which the last digit is 2.*

6) Dynamic Fork and Dynamic h*-out-of-*i *Join Pattern:* The dynamic fork and dynamic h-out-of-i join pattern integrates a dynamic fork construct (Pattern 1) with a dynamic h-out-of-i join construct (Pattern 8). The dynamic 1-out-of-i join construct, which is regarded as a dynamic discriminator, enables the subsequent activity when merely one trigger is received.

Pattern 14 (Dynamic Fork and Dynamic h-out-of-i Join)**.** *Suppose* P_0, P_1, P_2, Q_1, $\ldots$, Q_n, R_1, $\ldots$, $R_n \in S_{\mathrm{A}}$, $DF_1 \in S_{\mathrm{DF}}$, $DHIJ_1 \in S_{\mathrm{DHIJ}}$, (P_0, P_1), (P_1, DF_1), (DF_1, Q_k), $(R_k, DHIJ_1)$, $(DHIJ_1, P_2) \in S_{\mathrm{E}}$, $1 \le h \le n-1$, $2 \le i \le n$ *and* $h < i$ *for* $k = 1, \ldots, i$. *The activity* P_1 *determines the number of concurrent activities* i *at runtime. The dynamic fork* DF_1 *then splits the incoming flow from the activity* P_1 *into* i *outgoing flows to the concurrent activities* Q_1, $\ldots$, Q_i *in which all flows finally connect to the dynamic h-out-of-i join* $DHIJ_1$. *The dynamic h-out-of-i join* $DHIJ_1$ *activates the activity* P_2 *when* h *triggers are received from the activities* R_1, $\ldots$, R_i. *All other* $i - h$ *triggers received subsequently are ignored and* $DHIJ_1$ *resets itself.*

Definition 12. *Suppose a dynamic join and dynamic h-out-of-i join pattern,* $\psi_{\mathrm{E}}((P_0, P_1)) = p_1$, $\psi_{\mathrm{E}}((P_1, DF_1)) = df_1$, $\psi_{\mathrm{E}}((DF_1,\ Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k,\ DHIJ_1)) = dhij_{1,k}$, $\psi_{\mathrm{E}}((DHIJ_1,\ P_2)) = p_2$, $eval$ *is a channel for determining the number of alternative activities and* $val_2, \ldots, val_n$ *are channels for modelling the possible number of alternative activities,* $\psi_{\mathrm{A}}(P_1) = P_1(p_1, p_2,\ eval,\ val_2, \ldots, val_n,\ df_1,\ q_1, \ldots, q_n,\ dhij_{1,1}, \ldots, dhij_{1,n},\ \psi_{\mathrm{DF}}(DF_1) = DF_1(df_1,\ q_1, \ldots, q_i)$ *and* $\psi_{\mathrm{DHIJ}}(DHIJ_1) = DHIJ_1(dhij_{1,1},\ \ldots,\ dhij_{1,i},\ p_2)$. *The activity* P_1*, dynamic fork* DF_1 *and dynamic h-out-of-i join* $DHIJ_1$ *are specified as:*

$$\begin{aligned}
&P_1(p_1, p_2, eval, val_2, \ldots, val_n, df_1, q_1, \ldots, q_n,\\
&\quad dhij_{1,1}, \ldots, dhij_{1,n}) \stackrel{\mathrm{def}}{=}
\end{aligned}$$

$$(\boldsymbol{\nu} rs)p_1.\tau.\overline{eval}\langle rs\rangle.rs(x).$$
$$\sum_{i=2}^{n}[x = val_i](DF_1(df_1, q_1, \ldots, q_i)|$$
$$DHIJ_1(dhij_{1,1}, \ldots, dhij_{1,i}, p_2)|$$
$$\overline{df_1}.\mathbf{0})$$

$$DF_1(df_1, q_1, \ldots, q_i) \stackrel{\text{def}}{=}$$
$$df_1.\prod_{j=1}^{i}\overline{q_j}.\mathbf{0}$$

$$DHIJ_1(dhij_{1,1}, \ldots, dhij_{1,i}, p_2) \stackrel{\text{def}}{=}$$
$$(\boldsymbol{\nu} received)$$
$$(\prod_{j=1}^{i} dhij_{1,j}.\overline{received}.\mathbf{0}|$$
$$\underbrace{received.\cdots.received}_{h\ times}.\overline{p_2}.\underbrace{received.\cdots.received}_{i-h\ times}.$$
$$DHIJ_1(dhij_{1,1}, \ldots, dhij_{1,i}, p_2)).$$

The process $DHIJ_1$ is the π-calculus representation of the dynamic h-out-of-i join. A signal is sent along the channel p_2 after executing h times the input action $received$. The process $DHIJ_1$ then continues as itself after a further execution of $i-h$ times of the input action $received$.

The use of recursive definition rather than replication for modelling a dynamic h-out-of-i join construct is due to the fact that a reset is just like an invocation of itself which is recursive in nature.

Example 9. *The location of a retail shop and the availability of a particular product determine the number of other retail shops to be contacted whenever it runs out of stock for the product. A number of* ***contact other retail shop*** *activities is initiated in parallel. Once one of the activities* ***receive reply*** *is executed, the* ***confirm shipment*** *activity is activated and all other received replies are ignored.*

A workflow is a structured workflow [41] if (i) each fork construct is associated with a join construct; and (ii) each multi-choice is associated with a synchronizing merge. The dynamic fork and dynamic join pattern (Pattern 9) as well as the dynamic multi-choice and dynamic synchronizing merge pattern (Pattern 11) support the modelling of structured workflows. On the contrary, the dynamic fork and dynamic multi-merge pattern (Pattern 12), the dynamic multi-choice and dynamic multi-merge pattern (Pattern 13) as well as the dynamic fork and dynamic h-out-of-i join pattern (Pattern 14) allow for the modelling of unstructured workflows. Detailed discussions on the transformation of structured models into unstructured models are provided in [41] and [42].

In the following, we change the focus from dynamic workflow patterns that are built on the basic workflow patterns to those dynamic workflow patterns based on the structural relationships of activities and edges.

7) Dynamic Structured Loop Pattern: The principle behind the dynamic structured loop pattern is the number of activities between the entry and exit points is not necessary known at design time and can sometimes be only determined at runtime in accordance to various factors before the commencement of each iteration.

Pattern 15 (Dynamic Structured Loop)**.** *Suppose* P_0, P_1, P_2, P_3, Q_1, …, $Q_n \in S_{\mathrm{A}}$, $gc_1, gc_2 \in S_{\mathrm{GC}}$, (P_0, P_1), (P_1, Q_1), (Q_k, Q_{k+1}), (Q_i, P_2), (P_2, P_1), $(P_2, P_3) \in S_{\mathrm{E}}$, $\Phi_{\mathrm{GC}}((P_2, P_1)) = gc_1$, $\Phi_{\mathrm{GC}}((P_2, P_3)) = gc_2$ *and* $1 \leq i \leq n$ *for* $k = 1, \ldots, i-1$. *The activity* P_1 *is executed for determining the number of activities* i *to perform in sequence at runtime. After the completion of the activities* $Q_1, \ldots, Q_i$, *the thread of control is passed to the activity* P_2 *for evaluating the guard-conditions* gc_1 *and* gc_2. *The activities* P_1 *and* P_3 *are enabled, respectively, when* gc_1 *and* gc_2 *hold. The enablement of activity* P_1 *allows the repeated execution of the same or different number of activities* i *as the previous iteration of the dynamic structured loop.*

Definition 13. *Suppose a dynamic structured loop pattern,* $\psi_{\mathrm{E}}((P_0, P_1)) = p_1$, $\psi_{\mathrm{E}}((P_1, Q_1)) = q_1$, $\psi_{\mathrm{E}}((Q_k, Q_{k+1})) = q_{k+1}$, $\psi_{\mathrm{E}}((Q_i, P_2)) = p_2$, $\psi_{\mathrm{E}}((P_2, P_1)) = p_1$, $\psi_{\mathrm{E}}((P_2, P_3)) = p_3$, $eval$ *is a channel for determining the number of activities to perform in sequence,* $val_1, \ldots, val_n$ *are channels for modelling the possible number of activities to perform in sequence,* $\psi_{\mathrm{A}}(P_1) = P_1(p_1, p_2, eval, val_1, \ldots, val_n, q_1, \ldots, q_n)$, $\psi_{\mathrm{A}}(Q_m) = Q_m(q_{m,1}, q_{m,2})$ *and* $\psi_{\mathrm{A}}(P_2) = P_2(p_1, p_2, p_3, gc_1, gc_2)$. *The activities* P_1, P_2 *and* Q_m *for* $m = 1, \ldots, i$ *are specified by the following* π*-calculus specifications:*

$$P_1(p_1, p_2, eval, val_1, \ldots, val_n, q_1, \ldots, q_n) \stackrel{\text{def}}{=}$$
$$!(\boldsymbol{\nu} rs)p_1.\tau.\overline{eval}\langle rs\rangle.rs(x).$$
$$\sum_{i=1}^{n}[x = val_i](Q_1(q_1, q_2)|\ldots|Q_{i-1}(q_{i-1}, q_i)|$$
$$Q_i(q_i, p_2)|\overline{q_1}.\mathbf{0})$$

$$Q_m(q_{m,1}, q_{m,2}) \stackrel{\text{def}}{=}$$
$$!q_{m,1}.\tau.\overline{q_{m,2}}.\mathbf{0}$$

$$P_2(p_1, p_2, p_3, gc_1, gc_2) \stackrel{\text{def}}{=}$$
$$!p_2.\tau.(\boldsymbol{\nu} x_1, x_2, bval_1, bval_2)$$
$$((\prod_{j=1}^{2}\overline{gc_j}\langle x_j\rangle.x_j(y_j).\overline{bval_j}\langle y_j\rangle.\mathbf{0})|$$
$$bval_1(z_1).bval_2(z_2).$$
$$([z_1 = true]\overline{p_1}.\mathbf{0} +$$
$$[z_2 = true]\overline{p_3}.\mathbf{0})).$$

Unlike a dynamic h-out-of-i join, a dynamic structured loop is iterative in lieu of recursive. Processes P_1, Q_m and P_2 for $k = 1$ to i, which model the activities of a dynamic structured loop pattern, are expressed in the π-calculus through the use of replication.

Example 10. *Depending on the marital status of an employee, continue the* ***input personal particulars*** *activity or both the* ***input personal particulars*** *and* ***input spouse information*** *activities in sequential order until all new employees are processed.*

8) Dynamic Deferred Choice Pattern: The dynamic deferred choice provides an offer consisting of a number of alternative activities decided at runtime to the external environment for selection.

Pattern 16 (Dynamic Deferred Choice)**.** *Suppose* P_0, P_1 Q_1, …, Q_n, R_1, …, $R_n \in S_A$, (P_0, P_1), (P_1, Q_k), $(Q_k, R_k) \in S_{\mathrm{E}}$ *and* $2 \leq i \leq n$ *for* $k = 1, \ldots, i$. *The*

number of alternative activities i is not known at design time and is determined at the moment when the activity P_1 is executed. The choice of the alternative activities $Q_1, \ldots, Q_i$, which is connected to the activities $R_1, \ldots, R_i$, is deferred until the execution of one of the alternative activities begins. Unlike the dynamic exclusive choice, the decision is made by the environment instead of the activity P_1. The alternative activities which are not selected are then withdrawn.

Definition 14. *Suppose a dynamic deferred choice, $\psi_{\mathrm{E}}((P_0, P_1)) = p_1$, $\psi_{\mathrm{E}}((P_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((Q_k, R_k)) = r_k$, $eval$ is a channel for determining the number of alternative activities, $val_2, \ldots, val_n$ are channels for modelling the possible number of alternative activities, $q_1^{\mathrm{env}}, \ldots, q_n^{\mathrm{env}}$ are channels of $Q_1, \ldots, Q_n$ for interacting with the environment, $withdraw$ is a channel for withdrawal of non-selected alternative activities, $\psi_{\mathrm{A}}(P_1) = P_1(p_1, eval, val_2, \ldots, val_n, q_1, \ldots, q_n, q_1^{\mathrm{env}}, \ldots, q_n^{\mathrm{env}}, withdraw)$ and $\psi_{\mathrm{A}}(Q_j) = Q_j(q_j, q_j^{\mathrm{env}}, r_j, withdraw)$. The activities P_1 and Q_j for $j = 1, \ldots, i$ are denoted as:*

$$
\begin{aligned}
&P_1(p_1, eval, val_2, \ldots, val_n, q_1, \ldots, q_n, q_1^{\mathrm{env}}, \ldots, q_n^{\mathrm{env}}, \\
&\quad withdraw) \stackrel{\mathrm{def}}{=} \\
&\quad (\boldsymbol{\nu} rs) p_1.\tau.\overline{eval}\langle rs\rangle.rs(x). \\
&\quad \sum_{i=2}^{n}[x = val_i] \prod_{j=1}^{i}(Q_j(q_j, q_j^{\mathrm{env}}, r_j, withdraw)|\overline{q_j}.\mathbf{0}) \\
&Q_j(q_j, q_j^{\mathrm{env}}, r_j, withdraw) \stackrel{\mathrm{def}}{=} \\
&\quad q_j.(q_j^{\mathrm{env}}.\underbrace{\overline{withdraw}.\cdots.\overline{withdraw}}_{i-1 \ times}.\tau.\overline{r_j}.\mathbf{0} + \\
&\quad withdraw.\mathbf{0}).
\end{aligned}
$$

The input actions q_j^{env} for $j = 1$ to i model the selection performed by the environment. The execution of $i-1$ times of the output action $withdraw$ terminates all the alternative activities.

Example 11. *The number of specialists eligible for selection is based on the type of illness. The assignment of a specialist is deferred until one is available and all other **wait for a specialist** activities are withdrawn.*

9) Dynamic Interleaved Parallel Routing Pattern: The dynamic interleaved parallel routing pattern allows both the number of activities and their execution order to be determined dynamically.

Pattern 17 (Dynamic Interleaved Parallel Routing)**.** *Suppose $P_0, P_1, P_2, Q_1, \ldots, Q_n \in S_{\mathrm{A}}$, $(P_0, P_1), (Q_k, P_2) \in S_{\mathrm{E}}$ and $2 \leq i \leq n$ for $k = 1, \ldots, i$. The number of unordered activities i to be executed is determined at run time by the activity P_1. The activities $Q_1, \ldots, Q_i$ are executed sequentially without a particular order in which the activity P_2 is enabled after the completion of them.*

Definition 15. *Suppose a dynamic interleaved parallel routing pattern, $\psi_{\mathrm{E}}((P_0, P_1)) = p_1$, $\psi_{\mathrm{E}}((Q_k, P_2)) = p_2$, $eval$ is a channel for determining the number of unordered activities, $val_2, \ldots, val_n$ are channels for modelling the possible number of unordered activities, $\psi_{\mathrm{A}}(P_1) = P_1(p_1, p_2, eval, val_2, \ldots, val_n)$ and $\psi_{\mathrm{A}}(Q_k) = Q_k(execute, completed)$. The activities P_1 and Q_k are specified below:*

$$
\begin{aligned}
&P_1(p_1, p_2, eval, val_2, \ldots, val_n) \stackrel{\mathrm{def}}{=} \\
&\quad (\boldsymbol{\nu} rs) p_1.\tau.\overline{eval}\langle rs\rangle.rs(x). \\
&\quad \sum_{i=2}^{n}[x = val_i](\boldsymbol{\nu} execute, completed) \\
&\quad (\underbrace{\overline{execute}.completed.\cdots.\overline{execute}.completed}_{i \ times}.\overline{p_2}.\mathbf{0}| \\
&\quad \prod_{k=1}^{i} Q_k(execute, completed)) \\
&Q_k(execute, completed) \stackrel{\mathrm{def}}{=} \\
&\quad execute.\tau.\overline{completed}.\mathbf{0}.
\end{aligned}
$$

The sequence of output and input actions $\underbrace{\overline{execute}.completed.\cdots.\overline{execute}.completed}_{i \ times}$ randomly starts the execution of one of the activities and waits for the execution to complete before initiating another activity.

Example 12. *The total number of courses required to be taken for a semester is based on the amount of credits obtained previously. The enrollment of these courses can be performed sequentially in an arbitrary order.*

With a rigorous definition of the dynamic workflow patterns in terms of the π-calculus, software tools can then be utilized to visualize and simulate these patterns. Illustrative examples on the simulation of π-calculus specifications using PiVizTool are found in [43].

VI. Correctness of the Encodings

We now turn to the question of the validity of the encodings. To prove the encodings are correct, we need to introduce a concept of behavioural correspondence in which a pattern and the corresponding π-calculus representation exhibit the same behaviour. The strategy behind the proofs is to verify that each derived dynamic workflow pattern in Section V-B behaviourally corresponds to the transitions of its respective π-calculus specification. Before presenting the proofs, a description of the operational semantics of the π-calculus which is used primarily for constructing the proofs is given.

We start by letting $\Sigma^{\pi}_{\mathrm{IAct}} = \{x(\vec{y}) | x, \vec{y} \in \Sigma^{\pi}_{\mathrm{N}}\}$ and $\Sigma^{\pi}_{\mathrm{OAct}} = \{\overline{x}\langle\vec{y}\rangle | x, \vec{y} \in \Sigma^{\pi}_{\mathrm{N}}\}$ be the sets of input actions and output actions. We further assume that $\Sigma^{\pi}_{\mathrm{Act}} = \Sigma^{\pi}_{\mathrm{IAct}} \cup \Sigma^{\pi}_{\mathrm{OAct}} \cup \{\tau\}$ be the set of actions ranged over by α_i for $i = 1, \ldots, n$, $\Sigma^{\pi}_{\mathrm{MC}}$ be the set of matching constructs ranged over by N_i for $i = 1, \ldots, n$ and M be a finite match sequence $N_1 N_2 \ldots N_n$ where $n \geq 1$.

The operational semantics of the π-calculus is defined on the basis of either a reduction system as shown in Section 3 or a labelled transition system. The part of the operational semantics related to the proofs is formally captured by means of transitions as follows:

- $P \xrightarrow{\alpha} P'$: the execution of action α and process P becomes P'.
- $P \overset{M,\alpha}{\rightsquigarrow} P'$: if the match sequence M is true, action α is executed, if any, and process P becomes P'.

A complete treatment of the labelled transition semantics is beyond the scope of this section. We refer the reader to [33]

for more details. In what follows, we write $\xrightarrow{\alpha}$ and $\overset{M,\alpha}{\rightsquigarrow}$ as a shorthand for the transitions $P \xrightarrow{\alpha} P'$ and $P \overset{M,\alpha}{\rightsquigarrow} P'$ whenever only the action and match sequence are referred in our discussion.

Proposition 1. *Let P_1, P_2, Q_1, …, Q_n, R_1, …, $R_n \in S_{\mathrm{A}}$, $DF_1 \in S_{\mathrm{DF}}$, $DJ_1 \in S_{\mathrm{DJ}}$, (P_1, DF_1), (DF_1, Q_k), (R_k, DJ_1), $(DJ_1, P_2) \in S_{\mathrm{E}}$, $\psi_{\mathrm{E}}((P_1, DF_1)) = df_1$, $\psi_{\mathrm{E}}((DF_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k, DJ_1)) = dj_{1,k}$, $\psi_{\mathrm{E}}((DJ_1, P_2)) = p_2$, $eval$ is a channel for evaluating the number of concurrent activities and $2 \leq i \leq n$ for $k = 1, \ldots, i$. There is a behavioural correspondence between the dynamic fork and dynamic join pattern consisting of the activity P_1, dynamic fork DF_1 and dynamic join DJ_1 and the π-calculus specification.*

Proof sketch. *The execution of P_1 and evaluation of degree of parallelism are related to a sequence of transitions $\xrightarrow{\tau} \xrightarrow{\overline{eval}\langle rs\rangle} \xrightarrow{rs(x)}$ where rs is a channel created by P_1. The sending and receiving of control flows along (P_1, DF_1) correspond to transitions $\xrightarrow{\overline{df_1}}$ and $\xrightarrow{df_1}$. The multiple outgoing control flows of DF_1 are related to transitions $\xrightarrow{\overline{q_j}}$ for $j = 1$ to i. The receipt of control flows by DJ_1 and passing of control flow to P_2 behaviourally correspond to a sequence of transitions $\xrightarrow{\overline{dj_{1,j}}} \underbrace{\xrightarrow{\tau} \ldots \xrightarrow{\tau}}_{i \text{ times}} \xrightarrow{\overline{p_2}}$ for $j = 1$ to i. Thus, the behavioural correspondence holds.* □

Proposition 2. *Let P_1, P_2, Q_1, …, Q_n, R_1, …, $R_n \in S_{\mathrm{A}}$, $DEC_1 \in S_{\mathrm{DEC}}$, $DSM_1 \in S_{\mathrm{DSM}}$, (P_1, DEC_1), (DEC_1, Q_k), (R_k, DSM_1), $(DSM_1, P_2) \in S_{\mathrm{E}}$, $\psi_{\mathrm{E}}((P_1, DEC_1)) = dec_1$, $\psi_{\mathrm{E}}((DEC_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k, DSM_1)) = dsm_{1,k}$, $\psi_{\mathrm{E}}((DSM_1, P_2)) = p_2$, $eval$ is a channel for determining the number of alternative activities and $2 \leq i \leq n$ for $k = 1, \ldots, i$. There is a behavioural correspondence between the dynamic exclusive choice and dynamic simple merge pattern comprising the activity P_1, dynamic exclusive choice DEC_1 and dynamic simple merge DSM_1 and the π-calculus representation.*

Proof sketch. *We consider 2 cases.*

Case 1. We assume that the directed edges (DEC_1, Q_k) for $k = 1, \ldots, i$ are not associated with guard-conditions. The behaviour of P_1 corresponds to a sequence of transitions $\xrightarrow{\tau} \xrightarrow{\overline{eval}\langle rs\rangle} \xrightarrow{rs(x)}$ where rs is a new channel used for communication. The activation of DEC_1 is related to transitions $\xrightarrow{\overline{dec_1}}$ and $\xrightarrow{dec_1}$. The enablement of one of the outgoing edges behaviourally corresponds to one of the transitions $\xrightarrow{\overline{q_j}}$ for $j = 1, \ldots, i$. The incoming and outgoing control flows of DSM_1 correspond to a sequence of transitions $\xrightarrow{dsm_{1,j}} \xrightarrow{\overline{p_2}}$ for $j = 1, \ldots, i$. Thus, the behavioural correspondence holds.

Case 2. We assume that $gc_1, \ldots, gc_n \in S_{\mathrm{GC}}$, $|\, extract\text{-}GC(\{gc_1, \ldots, gc_i\}, true)\,| = 1$ and $\Phi_{\mathrm{GC}}((DEC_1, Q_k)) = gc_k$. The proof is the same as Case 1 with the exception that the evaluation of the guard-conditions $gc_1, \ldots, gc_i$ and the multiple outgoing control flows of DEC_1 correspond to sequences of transitions $\xrightarrow{\overline{gc_j}\langle x_j\rangle} \xrightarrow{x_j(y_j)} \underbrace{\xrightarrow{\tau} \ldots \xrightarrow{\tau}}_{i \text{ times}}$ and $\overset{[z_j = true], \overline{q_j}}{\rightsquigarrow}$ for $j = 1, \ldots, i$. □

Proposition 3. *Let P_1, P_2, Q_1, …, Q_n, R_1, …, $R_n \in S_{\mathrm{A}}$, $DMC_1 \in S_{\mathrm{DMC}}$, $DSynM_1 \in S_{\mathrm{DSynM}}$, (P_1, DMC_1), (DMC_1, Q_k), $(R_k, DSynM_1)$, $(DSynM_1, P_2) \in S_{\mathrm{E}}$, $\psi_{\mathrm{E}}((P_1, DMC_1)) = dmc_1$, $\psi_{\mathrm{E}}((DMC_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k, DSynM_1)) = dsynm_{1,k}$, $\psi_{\mathrm{E}}((DSynM_1, P_2)) = p_2$, $eval$ is a channel for determining the number of alternative activities and $2 \leq i \leq n$ for $k = 1, \ldots, i$. There is a behavioural correspondence between the dynamic multi-choice and dynamic synchronizing merge pattern which consists of the activity P_1, dynamic multi-choice DMC_1 and dynamic synchronizing merge $DSynM_1$ and the model in the π-calculus.*

Proof sketch. *Two cases are considered.*

Case 1. We assume that the directed edges (DMC_1, Q_k) for $k = 1, \ldots, i$ are not associated with guard-conditions. The proof is similar to Case 1 of Proposition 2.

Case 2. We assume that $gc_1, \ldots, gc_n \in S_{\mathrm{GC}}$, $|\, extract\text{-}GC(\{gc_1, \ldots, gc_i\}, true)\,| \geq 1$ and $\Phi_{\mathrm{GC}}((DMC_1, Q_k)) = gc_k$. The proof is analogous to Case 2 of Proposition 2. □

Proposition 4. *Let P_1, P_2, Q_1, …, Q_n, R_1, …, $R_n \in S_{\mathrm{A}}$, $DF_1 \in S_{\mathrm{DF}}$, $DMM_1 \in S_{\mathrm{DMM}}$, (P_1, DF_1), (DF_1, Q_k), (R_k, DMM_1), $(DMM_1, P_2) \in S_{\mathrm{E}}$, $\psi_{\mathrm{E}}((P_1, DF_1)) = df_1$, $\psi_{\mathrm{E}}((DF_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k, DMM_1)) = dmm_{1,k}$, $\psi_{\mathrm{E}}((DMM_1, P_2)) = p_2$, $eval$ is a channel for determining the number of concurrent activities and $2 \leq i \leq n$ for $k = 1, \ldots, i$. There is a behavioural correspondence between the dynamic fork and dynamic multi-merge pattern which comprises the activity P_1, dynamic fork DF_1 and dynamic multi-merge DMM_1 and the π-calculus implementation.*

Proof sketch. *Analogous to that of Proposition 1.* □

Proposition 5. *Let $P_1, P_2, Q_1, \ldots, Q_n, R_1, \ldots, R_n \in S_{\mathrm{A}}$, $DMC_1 \in S_{\mathrm{DMC}}$, $DMM_1 \in S_{\mathrm{DMM}}$, (P_1, DMC_1), (DMC_1, Q_k), (R_k, DMM_1), $(DMM_1, P_2) \in S_{\mathrm{E}}$, $\psi_{\mathrm{E}}((P_1, DMC_1)) = dmc_1$, $\psi_{\mathrm{E}}((DMC_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k, DMM_1)) = dmm_{1,k}$, $\psi_{\mathrm{E}}((DMM_1, P_2)) = p_2$, $eval$ is a channel for determining the number of alternative activities and $2 \leq i \leq n$ for $k = 1, \ldots, i$. There is a behavioural correspondence between the dynamic multi-choice and dynamic multi-merge pattern consisting of the activity P_1, dynamic multi-choice DMC_1 and dynamic multi-merge DMM_1 and the π-calculus representation.*

Proof sketch. *The proof is similar to the proof of Proposition 2.* □

Proposition 6. *Let P_1, P_2, Q_1, …, Q_n, R_1, …, $R_n \in S_{\mathrm{A}}$, $DF_1 \in S_{\mathrm{DF}}$, $DHIJ_1 \in S_{\mathrm{DHIJ}}$, (P_1, DF_1), (DF_1, Q_k), $(R_k, DHIJ_1)$, $(DHIJ_1, P_2) \in S_{\mathrm{E}}$, $\psi_{\mathrm{E}}((P_1, DF_1)) = df_1$, $\psi_{\mathrm{E}}((DF_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((R_k, DHIJ_1)) = dhij_{1,k}$, $\psi_{\mathrm{E}}((DHIJ_1, P_2)) = p_2$, $eval$ is a channel for determining the number of alternative activities, $1 \leq h \leq n-1$, $2 \leq i \leq n$ and $h < i$ for $k = 1, \ldots, i$. There is a behavioural correspondence between the dynamic fork and dynamic h-out-of-i join pattern which is composed of the activity P_1, dynamic fork DF_1 and dynamic h-out-of-i join $DHIJ_1$ and the π-calculus specification.*

Proof sketch. *Analogous to Proposition 1.* □

Proposition 7. *Let P_1, P_2, P_3, Q_1, …, $Q_n \in S_{\mathrm{A}}$, gc_1, $gc_2 \in S_{\mathrm{GC}}$, (P_1, Q_1), (Q_k, Q_{k+1}), (Q_i, P_2), (P_2, P_1), $(P_2, P_3) \in S_{\mathrm{E}}$, $\psi_{\mathrm{E}}((P_1, Q_1)) = q_1$, $\psi_{\mathrm{E}}((Q_k, Q_{k+1})) = q_{k+1}$, $\psi_{\mathrm{E}}((Q_i, P_2)) = p_2$, $\psi_{\mathrm{E}}((P_2, P_1)) = p_1$, $\psi_{\mathrm{E}}((P_2, P_3)) =$*

p_3, $eval$ is a channel for determining the number of activities to perform in sequence and $1 \leq i \leq n$ for $k = 1, \ldots, i-1$. There is a behavioural correspondence between the dynamic structured loop pattern and the the π-calculus representation.
Proof sketch. *By the same argument as Proposition 1.* □

Proposition 8. *Let P_1, Q_1, $\ldots$, Q_n, R_1, $\ldots$, $R_n \in S_{\mathrm{A}}$, (P_1, Q_k), $(Q_k, R_k) \in S_{\mathrm{E}}$, $\psi_{\mathrm{E}}((P_1, Q_k)) = q_k$, $\psi_{\mathrm{E}}((Q_k, R_k)) = r_k$, $eval$ is a channel for determining the number of alternative activities, q_1^{env}, $\ldots$, q_n^{env} are channels of Q_1, $\ldots$, Q_n for interacting with the environment, $withdraw$ is a channel for withdrawal of non-selected alternative activities and $2 \leq i \leq n$ for $k = 1, \ldots, i$. There is a behavioural correspondence between the dynamic deferred choice pattern and the π-calculus implementation.*
Proof sketch. *Analogous to Proposition 1.* □

Proposition 9. *Let P_1, P_2, Q_1, $\ldots$, $Q_n \in S_{\mathrm{A}}$, $(Q_k, P_2) \in S_{\mathrm{E}}$, $\psi_{\mathrm{E}}((Q_k, P_2)) = p_2$, $eval$ is a channel for determining the number of unordered activities and $2 \leq i \leq n$ for $k = 1, \ldots, i$. There is a behavioural correspondence between the dynamic interleaved parallel routing and the π-calculus specification.*
Proof sketch. *By the same argument as Proposition 1.* □

VII. Conclusions

Our work is characterized by the capability to capture the notion of dynamics in the context of workflow management. It is regarded as an extension to the works of van der Aalst et al. and Russell et al. This paper, along with other previous studies, offers collections of workflow patterns to be used in business process management.

A number of basic dynamic workflow patterns has been presented in our work. With this basic dynamic workflow patterns in place, we have further introduced a set of derived dynamic workflow patterns and defined their corresponding π-calculus representations. The main contributions of this work are:

(i) this is a first attempt to study workflow patterns whose structures can be dynamically reconfigured; and
(ii) the proposed patterns are encoded in the π-calculus in lieu of ordinary workflow languages for facilitating formal analysis using various software tools [44], [45], [46], [47].

The result of this study is not only of theoretical interest, but also has considerable practical benefits. It serves as a solid basis for the soundness verification of workflows constructed using dynamic workflow patterns. An additional interesting avenue of investigation is to explore how the dynamic workflow patterns can be expressed in terms of other formalisms.

References

[1] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, 2007.
[2] N. Russell, A. ter Hofstede, W. van der Aalst, and N. Mulyar, "Workflow control-flow patterns: A revised view," BPM Center, Tech. Rep. BPM-06-22, 2006, http://www.workflowpatterns.com/documentation/documents/BPM-06-22.pdf; accessed December 16, 2007.
[3] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, no. 3, pp. 5–51, 2003.
[4] OMG, "Business Process Model and Notation (BPMN), version 2.0," Jan. 2011, http://www.omg.org/spec/BPMN/2.0; accessed May 23, 2011.
[5] R. Milner, "The polyadic π-calculus: A tutorial," in *Logic and Algebra of Specification, Proceedings of International NATO Summer School*, vol. 94. Springer-Verlag, 1993, pp. 203–246.
[6] ——, *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
[7] V. Lam, "Dynamic workflow patterns," in *Proceedings of 2008 International Conference on Enterprise Information Systems and Web Technologies*. ISRST, 2008, pp. 160–166.
[8] B. Weber, S. Sadiq, and M. Reichert, "Beyond rigidity — dynamic process lifecyle support," *Computer Science: Research and Development*, vol. 23, no. 2, pp. 47–65, 2009.
[9] J. Mülle, K. Böhm, N. Röper, and T. Sünder, "Building conference proceedings requires adaptable workflow and content management," in *VLDB '06*, 2006, pp. 1129–1139.
[10] G. Decker, O. Kopp, F. Leymann, and M. Weske, "Interacting services: From specification to execution," *Data and Knowledge Engineering*, vol. 68, no. 10, pp. 946–972, 2009.
[11] G. Decker, O. Kopp, and F. Puhlmann, "Service referrals in BPEL-based choreographies," in *2nd European Young Researchers Workshop on Service Oriented Computing*, 2007, pp. 25–30.
[12] M. Pesic, M. Schonenberg, N. Sidorova, and W. van der Aalst, "Constraint-based workflow models: Change made easy," in *OTM 2007*, ser. LNCS 4803, 2007, pp. 77–94.
[13] W. van der Aalst, M. Pesic, and M. Schonenberg, "Declarative workflows: Balancing between flexibility and support," *Computer Science: Research and Development*, vol. 23, no. 2, pp. 99–113, 2009.
[14] A. Barros, G. Decker, M. Dumas, and F. Weber, "Correlation patterns in service-oriented architectures," in *FASE 2007*, ser. LNCS 4422, 2007, pp. 245–259.
[15] D. Yang and S. Zhang, "Approach for workflow modeling using π-calculus," *Journal of Zhejiang University SCIENCE*, vol. 4, no. 6, pp. 643–650, 2003.
[16] F. Puhlmann and M. Weske, "Using the π-calculus for formalizing workflow patterns," in *Business Process Management 2005*, 2005, pp. 153–168.
[17] G. Xue, J. Lu, and S. Yao, "Investigating workflow patterns in term of pi-calculus," in *11th International Conference on Computer Supported Cooperative Work in Design*. IEEE Computer Society, 2007, pp. 823–827.
[18] OMG, "Business process modeling notation specification," Feb. 2006, http://www.bpmn.org/; accessed December 28, 2007.
[19] D. Yang and S. Zhang, "Using π-calculus to formalize UML activity diagram for business process modeling," in *10th IEEE International Conference and Workshop on the Engineering of Computer-based Systems*. IEEE Computer Society, 2003, pp. 47–54.
[20] V. Lam, "On π-calculus semantics as a formal basis for UML activity diagrams," *International Journal of Software Engineering and Knowledge Engineering*, vol. 18, no. 4, pp. 541–567, 2008.
[21] P. Wohed, W. van der Aalst, M. Dumas, A. ter Hofstede, and N. Russell, "Pattern-based analysis of the control-flow perspective of UML activity diagrams," in *ER 2005*, 2005, pp. 63–78.
[22] N. Russell, W. van der Aalst, A. ter Hofstede, and P. Wohed, "On the suitability of UML 2.0 activity diagrams for business process modelling," in *APCCM 2006*, 2006, pp. 95–104.
[23] N. Russell, A. ter Hofstede, D. Edmond, and W. van der Aalst, "Workflow data patterns: Identification, representation and tool support," in *24th International Conference on Conceptual Modeling*, ser. LNCS 3716, 2005, pp. 353–368.
[24] N. Russell, W. van der Aalst, A. ter Hofstede, and D. Edmond, "Workflow resource patterns: Identification, representation and tool support," in *17th Conference on Advanced Information Systems Engineering*, ser. LNCS 3520, 2005, pp. 216–232.
[25] A. Barros, M. Dumas, and A. ter Hofstede, "Service interaction patterns," in *3rd International Conference on Business Process Management*, ser. LNCS 3649, 2005, pp. 302–318.
[26] A. Lanz, B. Weber, and M. Reichert, "Time patterns for process-aware information systems," *Requirements Engineering*, vol. 19, no. 2, pp. 113–141, 2014.
[27] F. Puhlmann, "Why do we actually need the pi-calculus for business process management?" in *BIS*, 2006, pp. 77–89.
[28] W. van der Aalst, "Pi calculus versus Petri nets: Let us eat "humble pie" rather than further inflate the "pi hype"," *BPTrends*, vol. 3, no. 5, pp. 1–11, 2005.
[29] V. Lam, "Recovering business process models with process patterns," in *Uncovering Essential Software Artifacts through Business Process Archeology*, R. Pérez-Castillo and M. Piattini, Eds. IGI Global, 2014, ch. 9, pp. 223–249.

[30] ——, "A formal execution semantics and rigorous analytical approach for communicating uml statechart diagrams." Ph.D. dissertation, University of Bath, 2006, also available as Technical Report, University of Bath, ISSN 1740-9497, 2006.
[31] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.
[32] D. Sangiorgi, "Expressing mobility in process algebras: First order and higher-order paradigms," Ph.D. dissertation, Computer Science Department, University of Edinburgh, 1993.
[33] J. Parrow, "An introduction to the π-calculus," in *Handbook of Process Algebra*, A. Bergstra, J.A. Ponse and S. Smolka, Eds. Elsevier Science, 2001, ch. 8, pp. 479–543.
[34] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
[35] T. Gschwind, J. Koehler, and J. Wong, "Applying patterns during business process modeling," in *BPM 2008*, ser. LNCS 5240, 2008, pp. 4–19.
[36] OMG, "UML 2.0 superstructure specification," Aug. 2005, http://www.omg.org; accessed July 28, 2006.
[37] G. Decker, F. Puhlmann, and M. Weske, "Formalizing service interactions," in *BPM 2006*, ser. LNCS 4102, 2006, pp. 414–419.
[38] H. Overdick, F. Puhlmann, and M. Weske, "Towards a formal model for agile service discovery and integration," in *ICSOC Workshop on Dynamic Web Processes*, 2005, http://www.icsoc.org; accessed June 18, 2011.
[39] M. Dumas, A. Grosskopf, T. Hettel, and M. Wynn, "Semantics of standard process models with OR-joins," in *OTM 2007*, ser. LNCS 4803, 2007, pp. 41–58.
[40] H. Völzer, "A new semantics for the inclusive converging gateway in safe processes," in *BPM 2010*, ser. LNCS 6336, 2010, pp. 294–309.
[41] B. Kiepuszewski, A. ter Hofstede, and C. Bussler, "On structured workflow modelling," in *CAiSE 2000*, ser. LNCS 1789, 2000, pp. 431–445.
[42] A. Polyvzanyy, L. García-Bañuelos, and M. Dumas, "Structuring acyclic process models," in *BPM 2010*, ser. LNCS 6336, 2010, pp. 276–293.
[43] A. Bog and F. Puhlmann, "A tool for the simulation of π-calculus systems," in *Open.BPM 2006: Geschäftsprozessmanagement mit Open Source-Technologien*, 2006.
[44] A. Bog, F. Puhlmann, and M. Weske, "The PiVizTool: Simulating choreographies with dynamic binding," in *Demo Session of the 5th International Conference on Business Process Management*, 2007, http://bpt.hpi.uni-potsdam.de/pub/Public/FrankPuhlmann/bpm2007-piviztool.pdf; accessed February 17, 2008.
[45] S. Briais, *The ABC User's Guide*, 2005, http://lamp.epfl.ch/~sbriais/abc/abc_ug.pdf; accessed February 17, 2008.
[46] U. Frendrup and J. Jensen, *A User Manual for the OBC Workbench*, Department of Computer Science, Aalborg University, 2001, http://www.cs.auc.dk/research/FS/ny/PR-pi; accessed January 20, 2005.
[47] B. Victor and F. Moller, "The mobility workbench: A tool for the π-calculus," in *CAV '94*, ser. LNCS 818, 1994, pp. 428–440.