Distributed Adaptation of Dining Philosophers

S. Andova[1], L.P.J. Groenewegen[2,⋆], and E.P. de Vink[1]

[1] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, The Netherlands

[2] FaST-Group, LIACS, Leiden University, The Netherlands

Abstract. Adaptation of a component-based system can be achieved in the coordination modelling language Paradigm through the special component McPal. McPal regulates the propagation of new behaviour and guides the changes in the components and in their coordination. Here we show how McPal may delegate part of its control to local adaptation managers, created on-the-fly, allowing for distribution of the adaptation indeed. We illustrate the approach for the well-known example of the dining philosophers problem, by modelling the migration from a deadlock-prone solution to a deadlock-free starvation-free solution without any system quiescence. The adaptation goes through various stages, exhibiting shifting control among McPal and its helpers, and changing degrees of orchestrated and choreographic collaboration.

1 Introduction

Many systems today are affected by changes in their operational environment when running, while they cannot be shutdown to be updated and restarted again. Instead, *dynamic adaptive systems* must be able to change their behaviour on-the-fly and to self-manage adaptation steps accommodating a new policy.

Dynamic adaptive systems consist of interacting components, usually distributed, and possibly hierarchically organized. In such a system, components may start adaptation in response to various triggers, such as changes in the underlying execution environment (e.g. failures or network congestion) or changes of requirements (e.g. imposed by the user). Adaptation of one component in the system may inadvertently influence the behavior of the components it is interacting with, possibly bringing about a cascade of dynamic changes in other parts of the system. Therefore, the adaptation of the system is a combination of local changes per component and global adaptation across components and hosts in the distributed system. As such, adaptation has to be performed in a consistent and coordinated manner so that the functionality of each separate component and of the system as a whole are preserved while the adaptation is in progress. Due to the complexity of the distributed dynamics of a system adapting on-the-fly, it may be rather difficult to understand whether a realization of a change plan indeed allows the system to perform as it is supposed to, and does not violate any of its requirements, during and after system adaptation.

⋆ Corresponding author, email: `luuk@liacs.nl`

One way to circumvent this is to formally model and analyze the system behaviour and the adaptation changes to be followed. In [13, 3, 6] we advocated how orchestrated adaptation can conveniently be captured in the coordination modeling language Paradigm. In Paradigm, a system architecture is organized along specific collaboration dimensions, called partitions. A partition is a well-chosen set of sub-behaviours of the local behaviour of a component, specifying the phases the component goes through when a protocol is executed. In the protocol, at a higher layer in the architecture, the component participates via its role, an abstract representation of the phases. A protocol manager coordinates the phase transfers for the components involved. In fact, in Paradigm, dynamic adaptation is modeled as just another collaboration protocol, coordinated by a special component *McPal* [13, 3, 6]. As progress within a phase is completely local to the component, the use of phase transfer instead of state transfer, is the key concept of Paradigm. This makes it possible to model, at the same time and separated from one another, both behavioural local changes per component, and global changes across architectural layers. The formal semantics of Paradigm then allows for a rigorous analysis of the adaptation policy [5], at the local level of the components as well as in the coordination of the changes across adaptive parts of the distributed system.

The suitability of Paradigm to model *distributed* adaptation strategies, extending our earlier centralized studies, is shown in this paper on a dining philosophers example. A deadlock-prone solution of the dining philosophers problem is taken as a source system, to be migrated to a target solution, both deadlock-free and starvation-free. Both systems are modeled in Paradigm, as is the migrating from source to target. As typical for dynamic adaptation in Paradigm, *McPal* regulates the propagation of new behaviour and guides the structural changes in the components and in their coordination. But here, although adding to the complexity of the solution, *McPal* delegates part of its control to local adaptation managers *McPhil*, one for each philosopher, while *McPal* keeps controlling them globally. Thus, we argue, the component-based character of the Paradigm language allows for modeling distributed adaptation: separate modeling of local strategies, coordinated by *McPal* as system adaptation manager. The main contribution of the paper is, it reveals the distributed potential of system adaptation within Paradigm. In Section 6 we elaborate on it.

Related work In recent years a number of approaches has been proposed addressing several issues of dynamic system adaptation. Some of them [14, 8, 18] focus on adaptive software architectures, where functionalities, considered as black boxes, are connected via ports. Formal modeling of dynamic adaptation has been addressed in e.g. [16, 2, 10, 22]. However, none of these approaches deal with distribution explicitly. In [11, 12] dynamic adaptation is formally modeled by means of graph transformation. Although graph transformation techniques are well suited for distributed systems, there is no explicit focus on modeling distributed control for adaptation in the papers mentioned.

Some aspects of dynamic adaptation of distributed systems, tailored for the domains considered, have been treated in [1, 17]. In the domain of Web Services, [1] proposes a method to generate distributed adapters from given service descriptions. [17] focuses on modeling and deployment of distributed resources for adaptive services in a mobile environment. A framework for formal modeling and verification of dynamic adaptation of distributed system, based on a transitional-invariant lattice technique, is proposed in [9]. The approach uses theorem proving techniques to show that during and after adaptation, the system always satisfies the transitional-invariants. This adaptation framework, however, does not support distribution in the style discussed in this paper: distributing adaptation tasks among local adaptation conductors by delegation.

The Conductor framework [21] for distributed adaptation allows for dynamic deployment of multiple adaptation conductors at various points in a network, an approach which is more suitable for complex and heterogeneous collaborations. It includes a distributed planning algorithm which determines for a triggered adaptation the most appropriate combination of conductors, distributed across the network. In [19] a distributed adaptation model for component-based applications is proposed. The model consists of two types of functionalities: mandatory that manage basic adaptation operations and optional that can be used to distribute adaptation activities. This way the adaptation mechanism of the whole system can be hierarchically organized, resembling as such our hierarchical structures of *McPal* conductors. However, both in [19] and [21], the main focus is on designing the adaptation itself, while the formal modeling and analysis of the adaptation remains uncovered, positioning them complementary to our treatment of distributed adaptation.

Structure of the paper Section 2 is an overview of Paradigm through the example of the deadlock-prone solution as source system. The target system, deadlock and starvation free, is in Section 3. Section 4 gives the distributed migration set-up from source to target system, with Section 5 discussing coordination technicalities separately. Section 6 wraps up and provides conclusions.

2 Dining philosophers as-is: deadlock-prone

This section presents a first solution to the dining philosopher problem of five Phil_i components sharing five Fork_i components, $i = 1..5$. We shall refer to this solution as the *as-is* system or just *as-is*. The solution itself is the well-known and failing deadlock-prone solution: Any Phil_i, while thinking and getting hungry, first waits until the left Fork_i can be got, then gets it, waits until the right Fork_{i+1} can be got, gets it and once having both forks starts eating. After the eating has satisfied her hunger, Phil_i lays down both forks and returns to thinking again. As an extra requirement, the as-is system has the ability to migrate from its ongoing as-is solution behaviour to to-be solution behaviour, unknown as yet but hopefully better than the failing as-is behaviour.

Apparently, steps taken by *Phil*s and step-like status changes of *Fork*s are to be consistently aligned in accordance to the particular as-is solution. This means, behaviour of the five *Phil*s and *Fork*s has to be coordinated such that the as-is solution is realized, failing as it may. Based on its capabilities for keeping ongoing behaviour constrained, the Paradigm language can specify coordination solutions not only for foreseen situations like the as-is system, but also for originally unforeseen migration to a still unknown to-be solution. Even more, Paradigm allows for really smooth migrations, i.e. with ongoing but gradually changing coordination during adaptation from as-is to to-be. In view thereof, one may have the special component *McPal* in a Paradigm model, at first not influencing the model at all, but hibernatingly present to guide upcoming system migration.

Through the example of the as-is solution for the five *Phil* and *Fork* components with a hibernating *McPal* in place, we shall briefly introduce Paradigm. The coordination modeling language Paradigm has five basic notions: STD, phase, (connecting) trap, role and consistency rule. For more elaborate introductions see [5] (in-depth) or [4] (more intuitive).

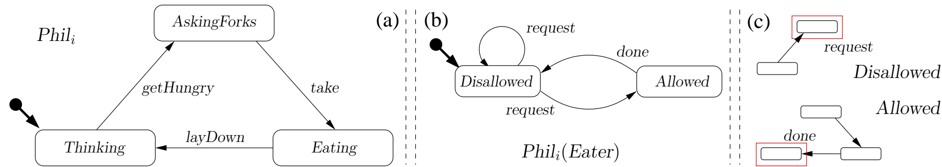

Fig. 1. The five $Phil_i$: (a) STD, (b) role *Eater*, (c) phase/trap constraints.

Component behaviour is specified by *STD*s, state-transition diagrams. Figure 1a gives the STD for each $Phil_i$ in UML style. It says, $Phil_i$ starts in state *Thinking* and has forever cycling behaviour, passing through her three states by repeatedly taking her three actions *getHungry*, *take*, *layDown* in that order. When $Phil_i$ gets hungry in *Thinking*, she takes action *getHungry*, thus arriving in *AskingForks*. In accordance to the as-is solution, when an arbitrary $Phil_i$ is sojourning in state *AskingForks*, the following is supposed to happen subsequently: (*i*) $Fork_i$ is claimed by her; (*ii*) $Fork_i$ is assigned to her; (*iii*) $Fork_{i+1}$ is claimed by her; (*iv*) $Fork_{i+1}$ is assigned to her. Thereupon $Phil_i$ performs action *take* for taking up the two *Fork*s assigned to her by now from the table, thus arriving in state *Eating*. Later when no longer hungry, $Phil_i$ goes from *Eating* to *Thinking* by taking action *layDown* for returning both *Fork*s to the table. We see, claiming and assigning of *Fork*s is not reflected in the STD steps of $Phil_i$.

In Paradigm, such claiming and assigning is to be modeled through temporary constraints on STD behaviour; here on $Fork_i$ and on $Fork_{i+1}$ behaviour influenced by $Phil_i$, as we shall see below. What we can observe already, also $Phil_i$'s STD behaviour is like-wise influenced, i.e. temporarily constrained, by the combined behaviours of $Fork_i$ and $Fork_{i+1}$, as $Phil_i$ can proceed to state *Eating* only if both *Fork*s have been assigned to her and remain so. In addition, as long as the *Fork*s remain assigned to her, $Phil_i$ can return to *Thinking* but she should not be able to proceed to *AskingForks* while holding them.

In general, within Paradigm a component participating in a collaboration does not contribute to the collaboration via its STD behaviour directly, but via a so-called *role*. Such a role is a different, global STD for the component built on top of the original STD, dealing with the temporary constraints that are important to the collaboration. The role contributes relevant essence only, role-wise distilled from the more detailed local component behaviour.

Figure 1b specifies STD role $\mathit{Phil}_i(\mathit{Eater})$ contributed by Phil_i to the collaboration called $\mathit{Phil2Forks}_i$. States of role $\mathit{Phil}_i(\mathit{Eater})$ are referred to as *phases* of the Phil_i STD: temporarily valid behavioural constraints imposed on Phil_i. Figure 1b mentions two phases: *Disallowed* and *Allowed*. Figure 1c graphically couples the two phases to Phil_i, by representing each phase as a subSTD, a scaled-down part of Phil_i. As one can see, phase *Disallowed* (on top) prohibits Phil_i to be in *Eating* but she may get as far as *AskingForks*. Contrarily, phase *Allowed* (at bottom) permits Phil_i to enter and to leave *Eating* once, but returning to *AskingForks* is not allowed.

Phase drawings are additionally decorated with one or more polygons, each polygon grouping states of that phase. In the simple case of Figure 1 polygons are rectangles comprising a single state. Polygons visualize so-called *traps*: a trap, as a subset of states in a phase, once entered, cannot be left as long as the phase remains the constraint imposed. A trap serves as a guard for a phase transfer (in role STDs). Therefore, traps label transitions in a role STD, cf. Figure 1b: the guard marking the transition from the previous phase (it is a trap of) to a next phase. In such a case, where all states in a trap are indeed states of the next phase, the trap is called *connecting* from the previous phase to the next.

Thus, role $\mathit{Phil}_i(\mathit{Eater})$ behaviour, see Figure 1b, expresses the ongoing alternation between *Disallowed* and *Allowed*: phase transfer from *Disallowed* to *Allowed* only happens after connecting trap *request* has been entered; similarly, phase transfer from *Allowed* to *Disallowed* only happens after connecting trap *done* has been entered. Moreover, an explicitly prolonged sojourn in *Disallowed* can happen after the (connecting) trap *request* has been entered.

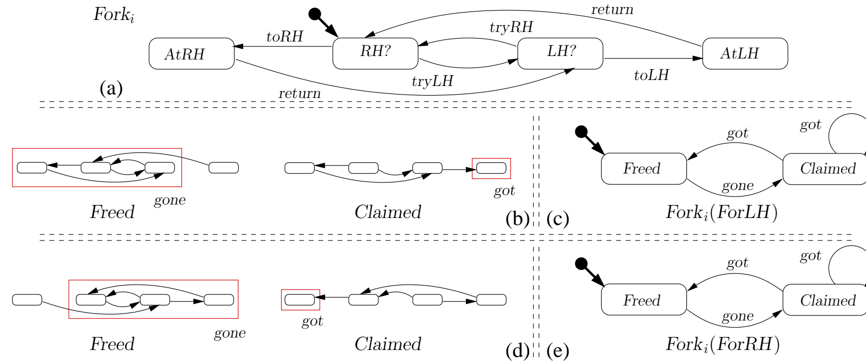

Fig. 2. Five Fork_i: (a) STD, (b,d) phase/trap constraints per (c,e) *ForLH*, *ForRH* role.

The STD $Fork_i$ is visualized in Figure 2a. State *AtRH* means, $Fork_i$ is assigned to (the right hand of) $Phil_{i-1}$. Similarly, state *AtLH* means, $Fork_i$ is assigned to (the left hand of) $Phil_i$. To express not being assigned to any of the two philosophers $Phil_{i-1}$ or $Phil_i$, STD $Fork_i$ is on the table, but in two different states *LH?* and *RH?*, reflecting a different bias: in *LH?* the bias is to $Phil_i$ and in *RH?* the bias is to $Phil_{i-1}$. In addition, upon returning to the table from having been assigned to a philosopher's hand, the bias is first to the other philosopher. This means, each *Fork* follows a round robin approach for honouring requests from $Phil_{i-1}$ and $Phil_i$, rather than a nondeterministic one. Figure 2 also presents two roles of $Fork_i$: (c) role *ForLH* for collaborating with $Phil_i$ (left hand) and (e) role *ForRH* for collaborating with $Phil_{i-1}$ (right hand). Role *ForLH* is based on phases *Freed* and *Claimed* and their connecting traps *gone* and *got* as given in part 2b. Note for instance, how in *Freed* of role *ForLH* the particular $Fork_i$ is being steered towards giving up staying assigned to $Phil_i$'s left hand, thus returning to the table with the possibility to get assigned to $Phil_{i-1}$'s right hand but, for the moment, not to $Phil_i$'s left hand.

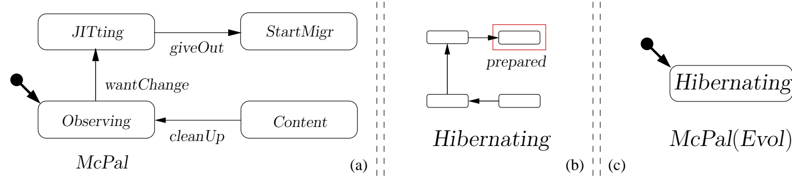

Fig. 3. *McPal*: (a) STD, (b) phase/trap constraint (b,d) role *McPal*(*Evol*).

The five *Phil*s and *Fork*s are all component ingredients needed for the as-is system. In view of still unknown later adaptation, an extra STD *McPal* is in place, see Figure 3a. Here *McPal* is in its hibernating form, not interfering at all with the as-is system, but with the ability to interfere with itself first, thus adapting itself and then later, as a consequence of its gained dynamics, to interfere with the as-is system. Figure 3bc underline this idea: (i) via phase *Hibernating* being the full *McPal* behaviour as long as *McPal* has not adapted itself yet; (ii) via role *Evol* which is restricted to sojourning in phase *Hibernating* as long as *McPal* remains unchanged. Thus we see, *McPal* starts in *Observing* and via *JITting* can go as far as *StartMigr*, which coincides with entering trap *prepared* of *Hibernating*. What cannot be seen from the figure but only from the consistency rules given below, through step *giveOut* leading into trap *prepared*, the hibernating *McPal* will extend the Paradigm as-is model specification with a specification of a to-be model as well as with a well-fitting model fragment for possible migration trajectories from as-is to to-be. To this aim, *McPal* embodies the reflectivity of a Paradigm model, by owning a local variable *Crs* where it stores the current model specification: consistency rules with all STDs, phases, traps and roles involved. Thus, by taking step *giveOut*, *Crs* will be extended, with at least one step series from *StartMigr* to *Content*, such that the no-longer-hibernating *McPal* is able to coordinate the various migration trajectories. Having returned to phase *Hibernating*, step *cleanUp* from *Content* to *Observing*

then refreshes *Crs* by removing all model fragments obsolete by then, keeping the to-be model only. Note, so far *McPal* is the same as in [3, 6].

In terms of the STDs, phases, traps and roles, Paradigm defines the 'coordination glue' between them through its notion of a *consistency rule*, being a synchronization of single role steps from different roles. Such a consistency rule may be coupled –additionally synchronized– with one detailed step of a so-called conductor component. Also local variables, such as *Crs*, can be updated. A consistency rule has as format: (*i*) it contains one asterisk $*$, with $*$'s right-hand side nonempty; (*ii*) optionally, at the left-hand side of $*$ it gives the one conductor step if relevant; (*iii*) at the right-hand side of $*$ it gives the listing of the role steps being synchronized; (*iv*) optionally, at the right-hand side $*$ a change clause can be given for updating variables. A consistency rule with a conductor step is called an *orchestration* step, a consistency rule without it is called a *choreography* step.

The set of consistency rules for the coordination of the as-is system, with *McPal* in place, is as follows.

$$* \;\; Phil_i(Eater)\colon Disallowed \overset{request}{\to} Disallowed,\;\; Fork_i(ForLH)\colon Freed \overset{gone}{\to} Claimed \tag{1}$$

$$* \;\; Fork_i(ForLH)\colon Claimed \overset{got}{\to} Claimed,\;\; Fork_{i+1}(ForRH)\colon Freed \overset{gone}{\to} Claimed \tag{2}$$

$$* \;\; Fork_{i+1}(ForRH)\colon Claimed \overset{got}{\to} Claimed,\;\; Phil_i(Eater)\colon Disallowed \overset{request}{\to} Allowed \tag{3}$$

$$\begin{aligned} * \;\; & Phil_i(Eater)\colon Allowed \overset{done}{\to} Disallowed, \\ & Fork_i(ForLH)\colon Claimed \overset{got}{\to} Freed,\;\; Fork_{i+1}(ForRH)\colon Claimed \overset{got}{\to} Freed \end{aligned} \tag{4}$$

$$McPal\colon JITting \overset{giveOut}{\to} StartMigr \;\; * \;\; McPal\colon [\, Crs := Crs + Crs_{migr} + Crs_{toBe} \,] \tag{5}$$

$$McPal\colon Content \overset{cleanUp}{\to} Observing \;\; * \;\; McPal\colon [\, Crs := Crs_{toBe} \,] \tag{6}$$

It is through consistency rules (1)–(4) the deadlock-prone solution is achieved. Their choreographic specification can be read like this (numbers referring to rules): (1) if $Phil_i$ wants to eat and her left $Fork_i$ hasn't been claimed yet, it is claimed; (2) if $Phil_i$ has got her left $Fork_i$ assigned and her right $Fork_{i+1}$ hasn't been claimed yet, it is claimed; (3) if $Phil_i$ has got her right $Fork_{i+1}$ assigned too, she is allowed to eat and can start doing so; (4) if $Phil_i$ stops eating, her $Fork_i$ and $Fork_{i+1}$ are being freed and she is prohibited to eat any longer. In addition, rules (5)–(6) are orchestration steps with *McPal* as conductor, not influencing ongoing collaborative as-is behaviour, but extending the as-is model specification (5) and reducing the model specification to the to-be specification aimed at (6), after the migration has been done. The migration itself is not specified here, as neither the to-be situation nor intermediate migration are known at present. Please note, *Crs* is a variable of *McPal*. Similarly, both Crs_{migr} and Crs_{toBe} are variables containing consistency rules too, which means, their final value will be determined in view of the particular migration trajectory traversed.

3 Dining philosophers to-be: no deadlock, no starvation

Before addressing migration in Sections 4 and 5, this section presents the goal to be reached by the migration, referred to as the *to-be* system or *to-be* solution. The

problem situation is the same as the one of the as-is situation, the five Phil_i and five Fork_i. But, the solution is better now: no deadlock and also no starvation. This is achieved in the following well-known way: for at least one Phil_i, but not for all five, the order of claiming Fork_i and Fork_{i+1} is being reversed.

For the Paradigm model of the to-be solution this means, all STDs, phases, traps and roles remain the same, but the consistency rules are different. For their formulation we need some extra notation. Let the index sets L, R be a non-empty disjoint partitioning of $\{1..5\}$, L referring to those Phil_is for which the left Fork_i is claimed first, and R referring to those Phil_is for which the right Fork_{i+1} is claimed first. Here we use $i \in L$, $j \in R$.

$$* \;\; \mathit{Phil}_i(\mathit{Eater})\colon \mathit{Disallowed} \stackrel{\mathit{request}}{\rightarrow} \mathit{Disallowed},\;\; \mathit{Fork}_i(\mathit{ForLH})\colon \mathit{Freed} \stackrel{\mathit{gone}}{\rightarrow} \mathit{Claimed} \tag{7}$$

$$* \;\; \mathit{Fork}_i(\mathit{ForLH})\colon \mathit{Claimed} \stackrel{\mathit{got}}{\rightarrow} \mathit{Claimed},\;\; \mathit{Fork}_{i+1}(\mathit{ForRH})\colon \mathit{Freed} \stackrel{\mathit{gone}}{\rightarrow} \mathit{Claimed} \tag{8}$$

$$* \;\; \mathit{Fork}_{i+1}(\mathit{ForRH})\colon \mathit{Claimed} \stackrel{\mathit{got}}{\rightarrow} \mathit{Claimed},\;\; \mathit{Phil}_i(\mathit{Eater})\colon \mathit{Disallowed} \stackrel{\mathit{request}}{\rightarrow} \mathit{Allowed} \tag{9}$$

$$* \;\; \mathit{Phil}_i(\mathit{Eater})\colon \mathit{Allowed} \stackrel{\mathit{done}}{\rightarrow} \mathit{Disallowed}, \\ \mathit{Fork}_i(\mathit{ForLH})\colon \mathit{Claimed} \stackrel{\mathit{got}}{\rightarrow} \mathit{Freed},\;\; \mathit{Fork}_{i+1}(\mathit{ForRH})\colon \mathit{Claimed} \stackrel{\mathit{got}}{\rightarrow} \mathit{Freed} \tag{10}$$

$$* \;\; \mathit{Phil}_j(\mathit{Eater})\colon \mathit{Disallowed} \stackrel{\mathit{request}}{\rightarrow} \mathit{Disallowed},\;\; \mathit{Fork}_{j+1}(\mathit{ForRH})\colon \mathit{Freed} \stackrel{\mathit{gone}}{\rightarrow} \mathit{Claimed} \tag{11}$$

$$* \;\; \mathit{Fork}_{j+1}(\mathit{ForRH})\colon \mathit{Claimed} \stackrel{\mathit{got}}{\rightarrow} \mathit{Claimed},\;\; \mathit{Fork}_j(\mathit{ForLH})\colon \mathit{Freed} \stackrel{\mathit{gone}}{\rightarrow} \mathit{Claimed} \tag{12}$$

$$* \;\; \mathit{Fork}_j(\mathit{ForLH})\colon \mathit{Claimed} \stackrel{\mathit{got}}{\rightarrow} \mathit{Claimed},\;\; \mathit{Phil}_j(\mathit{Eater})\colon \mathit{Disallowed} \stackrel{\mathit{request}}{\rightarrow} \mathit{Allowed} \tag{13}$$

$$* \;\; \mathit{Phil}_j(\mathit{Eater})\colon \mathit{Allowed} \stackrel{\mathit{done}}{\rightarrow} \mathit{Disallowed}, \\ \mathit{Fork}_j(\mathit{ForLH})\colon \mathit{Claimed} \stackrel{\mathit{got}}{\rightarrow} \mathit{Freed},\;\; \mathit{Fork}_{j+1}(\mathit{ForRH})\colon \mathit{Claimed} \stackrel{\mathit{got}}{\rightarrow} \mathit{Freed} \tag{14}$$

$$\mathit{McPal}\colon\; \mathit{JITting} \stackrel{\mathit{giveOut}}{\rightarrow} \mathit{StartMigr} \;*\; \mathit{McPal}\colon [\, \mathit{Crs} := \mathit{Crs} + \mathit{Crs}_{\mathit{migr}} + \mathit{Crs}_{\mathit{toBe}} \,] \tag{15}$$

$$\mathit{McPal}\colon \mathit{Content} \stackrel{\mathit{cleanUp}}{\rightarrow} \mathit{Observing} \;*\; \mathit{McPal}\colon [\, \mathit{Crs} := \mathit{Crs}_{\mathit{toBe}} \,] \tag{16}$$

Rules (7)–(10) together with (15)–(16) are exactly the rules (1)–(6) from Section 2. It is not difficult to observe, rules (11)–(14) mirror (7)–(10) by reversing the order of claiming indeed. Furthermore, note that only the consistency rules have been adapted, so the change remains restricted to the ‘coordination glue’ between the components, particularly the choreography steps only. *McPal* is in hibernation, as usual with no migration going.

4 Migration coordination set-up among helpers

As Section 3 announced, the migration to be realized is from the as-is situation to the to-be situation, i.e. starting from the deadlock-prone solution of the dining philosophers problem to the well-known, far better deadlock-free and starvation-free solution, where at least one Phil_i gets her Fork_i and Fork_{i+1} assigned in a different order. So, there is ample room for different to-be solutions meeting the requirements. Also, for each to-be solution different migration trajectories towards it can be developed.

In view of this observation, we restrict the range of our to-be solutions as follows: regarding the sets L, R introduced above –claiming left fork first for $Phil_i$ with $i \in L$ versus claiming right fork first for $Phil_j$ with $j \in R$– we require L and R to have either 2 or 3 elements. Moreover, if $L = \{i, i'\}$ then $Phil_i$ and $Phil_{i'}$ are not adjacent, i.e. $i = i' + 2$ or $i = i' + 3$, and similarly, if $R = \{j, j'\}$ then $Phil_j$ and $Phil_{j'}$ are not adjacent. Thus, for the to-be solution, of the two groups of *Phil*s one group consists of two *Phil*s and the other group consists of three *Phil*s. In addition, the *Phil*s from the group of two are not neighbours. This reduction in admitted to-be solutions will illustrate the interplay of central change management and local change management more clearly. Moreover, it helps us in substantially restricting the range of migration trajectories, still showing the dynamic flexibility of the migration[1].

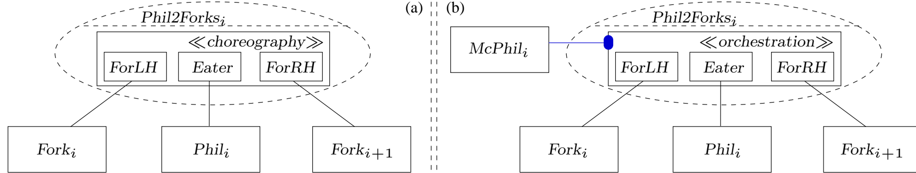

Fig. 4. Two collaboration snapshots (a) during hibernation, (b) during migration.

Before addressing the actual migration through coordination not yet specified, we want to clarify an important structural difference in the collaboration of $Phil_i$ and her two forks $Fork_i$ and $Fork_{i+1}$: during *McPal*'s hibernation versus during migration. Figure 4a, in UML-style, gives collaboration diagram $Phil2Forks_i$ during hibernation. It says, the three components $Phil_i$, $Fork_i$, $Fork_{i+1}$ are involved in it and, in line with the Paradigm model, they contribute to it via their respective roles *Eater*, *ForLH*, *ForRH*. This makes the collaboration into a choreography. Note, this architectural snapshot is valid for the as-is as well as for the to-be solution, the difference being in the behaviour only.

During the migration the collaboration has a slightly different structure, however, see Figure 4b. For each $Phil_i$ an extra component $McPhil_i$ is involved, meant as delegated helper of *McPal* for the $Phil2Forks_i$ collaboration only, to enlarge *McPal*'s influence. As we shall see below, $McPhil_i$ joins the collaboration as a new local driver of the ongoing choreography, thereby turning $Phil2Forks_i$ into an orchestration with $McPhil_i$ as its conductor, with essentially the same collaborative behaviour for a short while. Then, as conductor in place it migrates the orchestration and informs *McPal* about the result achieved so far, whereupon *McPal* decides about keeping or altering a result. Then $McPhil_i$ does so and steps back as conductor, turning collaboration $Phil2Forks_i$ into a choreography again.

McPal's actual migration activity is outlined in Figure 5: *McPal*, upon awakening from phase *Hibernating*, becomes active within phase *Migrating* as main conductor of the migration orchestration. Here it immediately delegates the actual migration coordination to its five helpers $McPhil_i$, by taking step *delegate*.

[1] For an animated migration trajectory, see the extended version of the FACS 2010 presentation at `http://www.win.tue.nl/~evink/research/paradigm.html`.

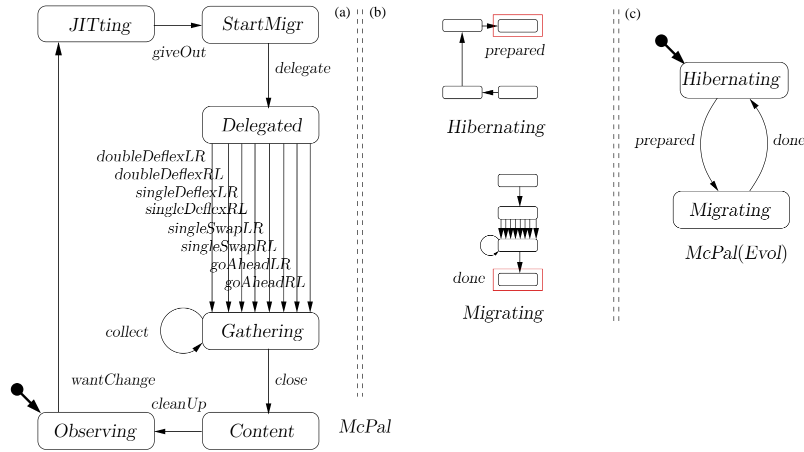

Fig. 5. *McPal* during migration: (a) STD, (b) phase/trap constraints, (c) role *Evol*.

In doing so, *McPal* provides each $McPhil_i$ with the orchestration rules for the local migration, while keeping the as-is choreography rules. Arrived in state *Delegated*, *McPal* then waits for the local results from the $McPhil_i$, being preliminary only. The preliminary results can be of two kinds: either $Phil_i$ (still) belongs to L or $Phil_i$ (now) belongs to R. Depending on the combined results of the five $McPhil_i$, conductor *McPal* takes one out of eight possible steps to state *Gathering*: by possibly letting zero, one or two specific $McPhil_i$ adjust their preliminary result when finalizing and by letting the other $McPhil_i$ make their preliminary results permanent. In state *Gathering*, *McPal* starts collecting the five sets $Crs_{i,toBe}$ of consistency rules the various $McPhil_i$ have to deliver when finalizing: the to-be choreography local to $Phil2Forks_i$, constituting $McPhil_i$'s final result. To this aim *McPal* takes step *collect* five times, one per $McPhil_i$. After having collected the five sets $Crs_{i,toBe}$ and after the five helper $McPhil_i$ have stopped their activities, *McPal* takes step *close* to state *Content*, thus entering trap *done* marking the final stage of the migration phase.

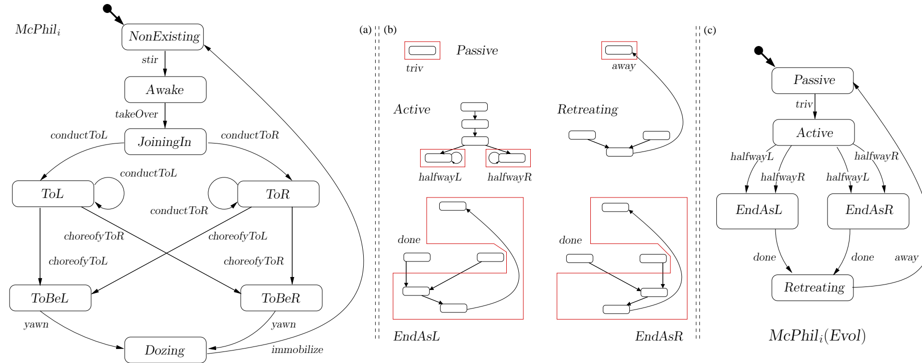

Fig. 6. $McPhil_i$ during migration: (a) STD, (b) phase/trap constraints, (c) role *Evol*.

The overall migration conducting of *McPal* sets the stage for the local migration exerted by $McPhil_i$ on the ongoing collaboration $Phil2Forks_i$. The behaviour of each $McPhil_i$ is drawn in Figure 6a. From starting state *NonExisting* to *Awake* it takes step *stir*, to get ready for whatever it has to do. From *Awake* it takes step *takeOver* to state *JoiningIn*, thereby removing the as-is choreography rules from the (local) model specification Crs_i, thus keeping the orchestration rules only, that were already added earlier by *McPal* when delegating the local migration to $McPhil_i$.

By taking step *conductToL* from state *JoiningIn* to state *ToL* and by iterating step *conductToL* in *ToL*, helper $McPhil_i$ sticks to the as-is orchestration, for the *L*-order that is. Similarly, by taking step *conductToR* from state *JoiningIn* to state *ToR* and by iterating step *conductToR* in *ToR*, helper $McPhil_i$ swaps the orchestration of the as-is choreography to the orchestration for the *R*-order. From state *ToL* helper $McPhil_i$ can, apart from iterating, either take step *choreofyToL* to state *ToBeL*, in which case $McPhil_i$ sticks to the *L*-order but turns the orchestration back into the equivalent choreography, or $McPhil_i$ can take step *choreofyToR* to state *ToBeR*, in which case $McPhil_i$ swaps to the *R*-order (on second thought, instigated by *McPal*) and moreover turns the orchestration into the equivalent choreography for the *R*-order. Analogously, from state *ToR* helper $McPhil_i$ can, apart from iterating, either take step *choreofyToR* to state *ToBeR*, in which case $McPhil_i$ sticks to the *R*-order but turns the orchestration into the equivalent choreography, or $McPhil_i$ can take step *choreofyToL* to state *ToBeL*, in which case $McPhil_i$ swaps back to the *L*-order and moreover turns the orchestration into the equivalent choreography for the *L*-order. From then on, in two consecutive steps, viz. *yawn* and *immobilize*, helper $McPhil_i$ returns to state *NonExisting*.

Figure 6b presents the phase and trap constraints on $McPhil_i$. Based on these constraints, role $McPhil_i(Evol)$ is given in part 6c. In phase *Passive* helper $McPhil_i$ can't do anything. In phase *Active* it can go as far as providing to *McPal* its preliminary result, being of two possible kinds, one per trap *halfwayL* or *halfwayR*. Phases *EndAsL* and *EndAsR* correspond to the two final results possible, the original *L*-order or the new *R*-order, respectively, available once trap *done* has been entered. Finally, in phase *Retreating* helper $McPhil_i$ enters trap *away*. After that it returns to *Passive* where it can't do anything.

It is stressed all this is to happen dynamically, on-the-fly, without any system halting. Consistency rules specifying this turn out to be quite technical. Therefore we discuss them separately in Section 5.

5 Migration coordination distributed among helpers

The consistency rules below specify the coordination according to Section 4's migration set-up. The technicalities of the rules mainly arise where change clauses manipulate sets of rules and model fragments aiming to influence the migration. Computing in terms of rules timely adapts the coordination strategy, gracefully enforcing the system's change. The following sets of consistency rules occur.

$$
\begin{array}{lll}
\mathit{Crs}_{i,asIs} & ::= & \text{choreography of } \mathit{Phil2Forks}_i,\ L\text{-order only (as in Section 2)} \\
\mathit{Crs}_{hibr} & ::= & \text{orchestration conducted by } \mathit{McPal} \text{ during phase } \mathit{Hibernating} \\
\mathit{Crs}_{noHb} & ::= & \text{orchestration conducted by } \mathit{McPal} \text{ during phase } \mathit{Migrating} \\
\mathit{Crs}_{i,orch} & ::= & \text{orchestration conducted by } \mathit{McPhil}_i \\
\mathit{Crs}_{i,toBeL} & ::= & \text{choreography of } \mathit{Phil2Forks}_i \text{ in } L\text{-order} \\
\mathit{Crs}_{i,toBeR} & ::= & \text{choreography of } \mathit{Phil2Forks}_i \text{ in } R\text{-order} \\
\mathit{Crs}_{migr} & ::= & \mathit{Crs}_{noHb} + \mathit{Crs}_{1,orch} + \cdots + \mathit{Crs}_{5,orch}
\end{array}
$$

The above sets are fixed, the sets below vary during the migration.

$$
\begin{array}{lll}
\mathit{Crs} & ::= & \text{varying orchestration/choreography, not governed by } \mathit{McPhil}_i \\
\mathit{Crs}_i & ::= & \text{varying } \mathit{McPhil}_i\text{-governed rule set for } \mathit{Phil2Forks}_i \\
\mathit{Crs}_{i,toBe} & ::= & \text{either } \mathit{Crs}_{i,toBeL} \text{ or } \mathit{Crs}_{i,toBeR} \\
\mathit{Crs}_{toBe} & ::= & \text{growing from } \mathit{Crs}_{hibr} \text{ to } \mathit{Crs}_{hibr} + \mathit{Crs}_{1,toBe} + \cdots + \mathit{Crs}_{5,toBe}
\end{array}
$$

The fixed sets of the consistency rules are specified first. Note, the variable sets of consistency rules are updated through detailed change clauses involving the fixed sets.[2] Consistency rules (17)–(20) making up $\mathit{Crs}_{i,asIs}$ are exactly rules (1)–(4) from Section 2.

$$* \ \mathit{Phil}_i(\mathit{Eater})\colon \mathit{Disallowed} \overset{\mathit{request}}{\to} \mathit{Disallowed},\ \mathit{Fork}_i(\mathit{ForLH})\colon \mathit{Freed} \overset{\mathit{gone}}{\to} \mathit{Claimed} \tag{17}$$

$$* \ \mathit{Fork}_i(\mathit{ForLH})\colon \mathit{Claimed} \overset{\mathit{got}}{\to} \mathit{Claimed},\ \mathit{Fork}_{i+1}(\mathit{ForRH})\colon \mathit{Freed} \overset{\mathit{gone}}{\to} \mathit{Claimed} \tag{18}$$

$$* \ \mathit{Fork}_{i+1}(\mathit{ForRH})\colon \mathit{Claimed} \overset{\mathit{got}}{\to} \mathit{Claimed},\ \mathit{Phil}_i(\mathit{Eater})\colon \mathit{Disallowed} \overset{\mathit{request}}{\to} \mathit{Allowed} \tag{19}$$

$$* \ \mathit{Phil}_i(\mathit{Eater})\colon \mathit{Allowed} \overset{\mathit{done}}{\to} \mathit{Disallowed},\ \mathit{Fork}_i(\mathit{ForLH})\colon \mathit{Claimed} \overset{\mathit{got}}{\to} \mathit{Freed},\ \mathit{Fork}_{i+1}(\mathit{ForRH})\colon \mathit{Claimed} \overset{\mathit{got}}{\to} \mathit{Freed} \tag{20}$$

Likewise, rules (21)–(22) making up Crs_{hibr}, are exactly rules (5)–(6) and also rules (15)–(16) from Section 2 and 3, respectively.

$$\mathit{McPal}\colon \mathit{JITting} \overset{\mathit{giveOut}}{\to} \mathit{StartMigr} \ * \ \mathit{McPal}\colon [\,\mathit{Crs} := \mathit{Crs} + \mathit{Crs}_{migr} + \mathit{Crs}_{toBe}\,] \tag{21}$$

$$\mathit{McPal}\colon \mathit{Content} \overset{\mathit{cleanUp}}{\to} \mathit{Observing} \ * \ \mathit{McPal}\colon [\,\mathit{Crs} := \mathit{Crs}_{toBe}\,] \tag{22}$$

Note the two assignments to Crs. In rule (21), on the verge of migration, Crs is extended with the rules in Crs_{migr} as well as in Crs_{toBe}, the latter set at this moment containing Crs_{hibr} only, already present in Crs. In rule (22), right after the migration, Crs is replaced by Crs_{toBe}, by then containing all choreography rules computed by the five *McPhil* plus the two rules in Crs_{hibr} already present.

Next we present the consistency rule set Crs_{noHb}, rules (23) to (36), covering the interaction of *McPal* and its five helper McPhil_i.

$$* \ \mathit{McPal}(\mathit{Evol})\colon \mathit{Hibernating} \overset{\mathit{prepared}}{\to} \mathit{Migrating} \tag{23}$$

$$\begin{aligned} &\mathit{McPal}\colon \mathit{StartMigr} \overset{\mathit{create}}{\to} \mathit{Delegated} \ * \ \mathit{McPal}\colon [\,\mathit{Crs} := \mathit{Crs}_{noHb} + \mathit{Crs}_{hibr}\,], \\ &\quad \mathit{McPhil}_1(\mathit{Evol})\colon \mathit{Passive} \overset{\mathit{triv}}{\to} \mathit{Active}, \ldots, \mathit{McPhil}_5(\mathit{Evol})\colon \mathit{Passive} \overset{\mathit{triv}}{\to} \mathit{Active}, \\ &\quad \mathit{McPhil}_1[\,\mathit{Crs}_1 := \mathit{Crs}_{1,asIs} + \mathit{Crs}_{1,orch}\,],\ \ldots,\ \mathit{McPhil}_5[\,\mathit{Crs}_5 := \mathit{Crs}_{5,asIs} + \mathit{Crs}_{5,orch}\,] \end{aligned} \tag{24}$$

[2] One may call this behaviour computation, programming in terms of behavioural constraints.

The set Crs_{noHb} contains the rule (23) for *McPal*'s own phase transfer from *Hibernating* to initiate the migration. From then on one finds orchestration rules for various conducting steps *McPal* may take within phase *Migrating*. Rule (24) gets the five helper $McPhil_i$ going, providing each with the local as-is choreographic rules as well as with its own orchestration rules, while *McPal* keeps those from Crs_{hibr} and Crs_{noHb} as its own rules only.

The STD of *McPal* in Figure 5 provides eight transitions from state *Delegated* to state *Gathering*. *McPal* takes a transition from its state *Delegated* to the state *Gathering* once all five $McPhil_i$ have reached a 'halfway' trap, either trap *halfwayL* or trap *halfwayR*, in their phase *Active*. Therefore, the figure shows eight different actions, dependent on the various combinations. By coupling a local transition of *McPal* to a global step in the *Evol* role of the $McPhil_i$, the proper transition is taken by *McPal* and the right continuation for the $McPhil_i$ is prescribed. Below, in the set of consistency rules Crs_{noHb}, we only provide the rules for the actions *doubleDeflexLR* and *singleDeflexRL*, rules (25) and (26), leaving the details of the remaining six rules to the reader.

$$\begin{aligned} &McPal\colon Delegated \stackrel{doubleDeflexLR}{\rightarrow} Gathering \;* \\ &\quad McPhil_1(Evol)\colon Active \stackrel{halfwayR}{\rightarrow} EndAsL,\; McPhil_2(Evol)\colon Active \stackrel{halfwayR}{\rightarrow} EndAsR, \\ &\quad McPhil_3(Evol)\colon Active \stackrel{halfwayR}{\rightarrow} EndAsL,\; McPhil_4(Evol)\colon Active \stackrel{halfwayR}{\rightarrow} EndAsR, \\ &\quad McPhil_5(Evol)\colon Active \stackrel{halfwayR}{\rightarrow} EndAsR \end{aligned} \tag{25}$$

$$\begin{aligned} &McPal\colon Delegated \stackrel{singleDeflexRL}{\rightarrow} Gathering \;* \\ &\quad McPhil_i(Evol)\colon Active \stackrel{halfwayR}{\rightarrow} EndAsR,\; McPhil_{i+1}(Evol)\colon Active \stackrel{halfwayL}{\rightarrow} EndAsL, \\ &\quad McPhil_{i+2}(Evol)\colon Active \stackrel{halfwayL}{\rightarrow} EndAsR,\; McPhil_{i+3}(Evol)\colon Active \stackrel{halfwayL}{\rightarrow} EndAsL, \\ &\quad McPhil_{i+4}(Evol)\colon Active \stackrel{halfwayL}{\rightarrow} EndAsL \end{aligned} \tag{26}$$

six more rules similar to (25) and (26) (27)–(32)

Rules (25)–(32) deal with the eight scenarios for handling the combined preliminary results from the five helper $McPhil_i$. In particular, rule (25) for action *doubleDeflexLR* covers the case where all five $McPhil_i$ follow *R*-order, so two non-neighbouring ones of them have to be swapped (back) to *L*-order, here we choose the first and the third. Similarly, rule (26) covers the case where exactly one $McPhil_i$ follows *R*-order, so another non-neighbouring one has to be swapped to *R*-order (as yet), here we choose $McPhil_{i+2}$.

$$\begin{aligned} &McPal\colon Gathering \stackrel{collect}{\rightarrow} Gathering \;*\; McPhil_i(Evol)\colon EndAsR \stackrel{done}{\rightarrow} Retreating, \\ &\quad McPal[\, Crs := Crs + Crs_i,\, Crs_{toBe} := Crs_{toBe} + Crs_{i,toBe}\,] \end{aligned} \tag{33}$$

$$\begin{aligned} &McPal\colon Gathering \stackrel{collect}{\rightarrow} Gathering \;*\; McPhil_i(Evol)\colon EndAsL \stackrel{done}{\rightarrow} Retreating, \\ &\quad McPal[\, Crs := Crs + Crs_i,\, Crs_{toBe} := Crs_{toBe} + Crs_{i,toBe}\,] \end{aligned} \tag{34}$$

$$\begin{aligned} &McPal\colon Gathering \stackrel{close}{\rightarrow} Content \;* \\ &\quad McPhil_1(Evol)\colon Retreating \stackrel{away}{\rightarrow} Passive,\; \ldots,\; McPhil_5(Evol)\colon Retreating \stackrel{away}{\rightarrow} Passive \end{aligned} \tag{35}$$

$$*\; McPal(Evol)\colon Migrating \stackrel{done}{\rightarrow} Hibernating \tag{36}$$

Rules (33) and (34) incorporate the final local R-order or the final local L-order, respectively, as final choreography part into the two variable sets Crs and Crs_{toBe}. Rule (35) passivates the five helper $McPhil_i$. Rule (36) allows $McPal$ to return into hibernation, mirroring rule (23).

The set $Crs_{i,orch}$ with the actual adaptation orchestration by $McPhil_i$, comprising the rules (37) to (68) below, follows the STD of Figure 6.

$$McPhil_i: Awake \stackrel{takeOver}{\rightarrow} JoiningIn \ * \ McPhil_i[\,Crs_i := Crs_i - Crs_{i,asIs}\,] \tag{37}$$

$$McPhil_i: JoiningIn \stackrel{conductToL}{\rightarrow} ToL \ * \quad Phil_i(Eater): Disallowed \stackrel{request}{\rightarrow} Disallowed,\ Fork_i(ForLH): Freed \stackrel{gone}{\rightarrow} Claimed \tag{38}$$

$$McPhil_i: JoiningIn \stackrel{conductToL}{\rightarrow} ToL \ * \quad Fork_i(ForLH): Claimed \stackrel{got}{\rightarrow} Claimed,\ Fork_{i+1}(ForRH): Freed \stackrel{gone}{\rightarrow} Claimed \tag{39}$$

$$McPhil_i: JoiningIn \stackrel{conductToR}{\rightarrow} ToR \ * \quad Phil_i(Eater): Disallowed \stackrel{request}{\rightarrow} Allowed,\ Fork_{i+1}(ForRH): Claimed \stackrel{got}{\rightarrow} Claimed \tag{40}$$

$$McPhil_i: JoiningIn \stackrel{conductToR}{\rightarrow} ToR \ * \ Phil_i(Eater): Allowed \stackrel{done}{\rightarrow} Disallowed, \quad Fork_i(ForLH): Claimed \stackrel{got}{\rightarrow} Freed,\ Fork_{i+1}(ForRH): Claimed \stackrel{got}{\rightarrow} Freed \tag{41}$$

$$McPhil_i: JoiningIn \stackrel{conductToR}{\rightarrow} ToR \ * \ Phil_i(Eater): Disallowed \stackrel{request}{\rightarrow} Disallowed, \quad Fork_i(ForLH): Claimed \stackrel{got}{\rightarrow} Freed,\ Fork_{i+1}(ForRH): Claimed \stackrel{triv}{\rightarrow} Claimed \tag{42}$$

Rule (37) removes the as-is choreography. Here, (38)–(41), with $McPhil_i$ in *JoiningIn*, cover the four previous choreography steps of the as-is protocol, cf. rules (17–20), but now orchestrated. Rules (38) and (39) lead the conductor to state *ToL* to continue conducting the original L-order; rules (40) and (41) lead the conductor to state *ToR* to continue according to the new R-order. In these two steps the swap from L-order to R-order is easy as it happens to coincide with stopping to eat or with getting hungry anew. Rule (42) is needed to escape deadlock, a subtlety not further elaborated here.

$$McPhil_i: ToR \stackrel{conductToR}{\rightarrow} ToR \ * \quad Phil_i(Eater): Disallowed \stackrel{request}{\rightarrow} Disallowed,\ Fork_{i+1}(ForRH): Freed \stackrel{gone}{\rightarrow} Claimed \tag{43}$$

$$McPhil_i: ToR \stackrel{conductToR}{\rightarrow} ToR \ * \quad Fork_{i+1}(ForRH): Claimed \stackrel{got}{\rightarrow} Claimed,\ Fork_i(ForLH): Freed \stackrel{gone}{\rightarrow} Claimed \tag{44}$$

$$McPhil_i: ToR \stackrel{conductToR}{\rightarrow} ToR \ * \quad Fork_i(ForLH): Claimed \stackrel{got}{\rightarrow} Claimed,\ Phil_i(Eater): Disallowed \stackrel{request}{\rightarrow} Allowed \tag{45}$$

$$McPhil_i: ToR \stackrel{conductToR}{\rightarrow} ToR \ * \ Phil_i(Eater): Allowed \stackrel{done}{\rightarrow} Disallowed, \quad Fork_i(ForLH): Claimed \stackrel{got}{\rightarrow} Freed,\ Fork_{i+1}(ForRH): Claimed \stackrel{got}{\rightarrow} Freed \tag{46}$$

four similar rules for cycling in *ToL* (47)–(50)

Rules (43)–(46), with $McPhil_i$ in *ToR*, cover the new R-order, basically implementing the to-be rules (11)–(14), but conducted by $McPhil_i$ while sojourning

in state *ToR*, waiting for *McPal*'s consent. The symmetric rules (47)–(50), with $McPhil_i$ staying in *ToL* are suppressed.

$$McPhil_i\colon ToL \stackrel{choreofyToL}{\rightarrow} ToBeL \ * \tag{51}$$
$$Phil_i(Eater)\colon Disallowed \stackrel{request}{\rightarrow} Disallowed,\ \ Fork_i(ForLH)\colon Freed \stackrel{gone}{\rightarrow} Claimed,$$
$$McPhil_i\ [\, Crs_i := Crs_i - Crs_{i,orch} + Crs_{i,toBeL},\ Crs_{i,toBe} := Crs_{i,toBeL}\,]$$

$$McPhil_i\colon ToL \stackrel{choreofyToL}{\rightarrow} ToBeL \ * \tag{52}$$
$$Fork_i(ForLH)\colon Claimed \stackrel{got}{\rightarrow} Claimed,\ \ Fork_{i+1}(ForRH)\colon Freed \stackrel{gone}{\rightarrow} Claimed,$$
$$McPhil_i\ [\, Crs_i := Crs_i - Crs_{i,orch} + Crs_{i,toBeL},\ Crs_{i,toBe} := Crs_{i,toBeL}\,]$$

$$McPhil_i\colon ToL \stackrel{choreofyToL}{\rightarrow} ToBeL \ * \tag{53}$$
$$Fork_{i+1}(ForRH)\colon Claimed \stackrel{got}{\rightarrow} Claimed,\ \ Phil_i(Eater)\colon Disallowed \stackrel{request}{\rightarrow} Allowed,$$
$$McPhil_i\ [\, Crs_i := Crs_i - Crs_{i,orch} + Crs_{i,toBeL},\ Crs_{i,toBe} := Crs_{i,toBeL}\,]$$

$$McPhil_i\colon ToL \stackrel{choreofyToL}{\rightarrow} ToBeL \ * \ Phil_i(Eater)\colon Allowed \stackrel{done}{\rightarrow} Disallowed, \tag{54}$$
$$Fork_i(ForLH)\colon Claimed \stackrel{got}{\rightarrow} Freed,\ \ Fork_{i+1}(ForRH)\colon Claimed \stackrel{got}{\rightarrow} Freed,$$
$$McPhil_i[\, Crs_i := Crs_i - Crs_{i,orch} + Crs_{i,toBeL},\ Crs_{i,toBe} := Crs_{i,toBeL}\,]$$

Rules (51)–(54), with $McPhil_i$ moving from *ToL* to *ToBeL*, cover the installment of *L*-order conducting as the to-be protocol. In addition, all orchestration in Crs_i is replaced by the *L*-order choreography.

$$McPhil_i\colon ToL \stackrel{choreofyToR}{\rightarrow} ToBeR \ * \ Phil_i(Eater)\colon Disallowed \stackrel{request}{\rightarrow} Disallowed, \tag{55}$$
$$Fork_i(ForLH)\colon Freed \stackrel{triv}{\rightarrow} Freed,\ \ Fork_{i+1}(ForRH)\colon Freed \stackrel{gone}{\rightarrow} Claimed,$$
$$McPhil_i[\, Crs_i := Crs_i - Crs_{i,orch} + Crs_{i,toBeR}, Crs_{i,toBe} := Crs_{i,toBeR}\,]$$

$$McPhil_i\colon ToL \stackrel{choreofyToR}{\rightarrow} ToBeR \ * \tag{56}$$
$$Fork_i(ForLH)\colon Claimed \stackrel{triv}{\rightarrow} Freed,\ \ Fork_{i+1}(ForRH)\colon Freed \stackrel{gone}{\rightarrow} Claimed,$$
$$McPhil_i[\, Crs_i := Crs_i - Crs_{i,orch} + Crs_{i,toBeR},\ Crs_{i,toBe} := Crs_{i,toBeR}\,]$$

$$McPhil_i\colon ToL \stackrel{choreofyToR}{\rightarrow} ToBeR \ * \quad Phil_i(Eater)\colon Disallowed \stackrel{triv}{\rightarrow} Disallowed, \tag{57}$$
$$Fork_i(ForLH)\colon Claimed \stackrel{triv}{\rightarrow} Freed,\ \ Fork_{i+1}(ForRH)\colon Claimed \stackrel{triv}{\rightarrow} Claimed,$$
$$McPhil_i\colon [Crs_i := Crs_i - Crs_{i,orch} + Crs_{i,toBeR},\ Crs_{i,toBe} := Crs_{i,toBeR}\,]$$

$$McPhil_i\colon ToL \stackrel{choreofyToR}{\rightarrow} ToBeR \ * \tag{58}$$
$$Phil_i(Eater)\colon Disallowed \stackrel{request}{\rightarrow} Allowed,\ \ Fork_{i+1}(ForRH)\colon Claimed \stackrel{got}{\rightarrow} Claimed,$$
$$McPhil_i[\, Crs_i := Crs_i - Crs_{i,orch} + Crs_{i,toBeR},\ Crs_{i,toBe} := Crs_{i,toBeR}\,]$$

$$McPhil_i\colon ToL \stackrel{choreofyToR}{\rightarrow} ToBeR \ * \ Phil_i(Eater)\colon Allowed \stackrel{done}{\rightarrow} Disallowed, \tag{59}$$
$$Fork_i(ForLH)\colon Claimed \stackrel{got}{\rightarrow} Freed,\ \ Fork_{i+1}(ForRH)\colon Claimed \stackrel{got}{\rightarrow} Freed,$$
$$McPhil_i[\, Crs_i := Crs_i - Crs_{i,orch} + Crs_{i,toBeR},\ Crs_{i,toBe} := Crs_{i,toBeR}\,]$$

nine rules for leaving *ToR*, similar to rules (51)–(59) (60)–(68)

Rules (55)–(59), with $McPhil_i$ heading for *ToBeR*, cover the orchestrated swapping from *L*-order to *R*-order, thereby installing it as choreography. In particular, (55) covers claiming the first (right) *Fork* without having to undo an earlier claim of the left *Fork*. Contrarily, (56) covers claiming the first (right) *Fork* together

with necessary undoing of an earlier claim of the left *Fork*. Notably, rule (57) covers continuing to claim the first (right) *Fork* together with necessary undoing of an earlier claim of the left *Fork*. Thus, (57) provides an escape from deadlock, similar to rule (42). Rule (58) and (59) cover starting and stopping to eat, for the last time as resulting from *L*-order. The symmetric rules (60)–(68) for leaving *ToR* are omitted.

Finally, the rule sets $Crs_{i,toBeL}$ and $Crs_{i,toBeR}$ contain the choreography rules for the to-be-situation in *L*-order and in *R*-order, rules (69)–(72) and (73)–(76) respectively.

$$* \ \ Phil_i(Eater): Disallowed \stackrel{request}{\to} Disallowed,\ \ Fork_i(ForLH): Freed \stackrel{gone}{\to} Claimed \tag{69}$$

$$* \ \ Fork_i(ForLH): Claimed \stackrel{got}{\to} Claimed,\ \ Fork_{i+1}(ForRH): Freed \stackrel{gone}{\to} Claimed \tag{70}$$

$$* \ \ Fork_{i+1}(ForRH): Claimed \stackrel{got}{\to} Claimed,\ \ Phil_i(Eater): Disallowed \stackrel{request}{\to} Allowed \tag{71}$$

$$* \ \ Phil_i(Eater): Allowed \stackrel{done}{\to} Disallowed,\\ Fork_i(ForLH): Claimed \stackrel{got}{\to} Freed,\ \ Fork_{i+1}(ForRH): Claimed \stackrel{got}{\to} Freed \tag{72}$$

$$* \ \ Phil_i(Eater): Disallowed \stackrel{request}{\to} Disallowed,\ \ Fork_{i+1}(ForRH): Freed \stackrel{gone}{\to} Claimed \tag{73}$$

$$* \ \ Fork_{i+1}(ForRH): Claimed \stackrel{got}{\to} Claimed,\ \ Fork_i(ForLH): Freed \stackrel{gone}{\to} Claimed \tag{74}$$

$$* \ \ Fork_i(ForLH): Claimed \stackrel{got}{\to} Claimed,\ \ Phil_i(Eater): Disallowed \stackrel{request}{\to} Allowed \tag{75}$$

$$* \ \ Phil_i(Eater): Allowed \stackrel{done}{\to} Disallowed,\\ Fork_i(ForLH): Claimed \stackrel{got}{\to} Freed,\ \ Fork_{i+1}(ForRH): Claimed \stackrel{got}{\to} Freed \tag{76}$$

Note, rules (17)–(76) cover all migration trajectories. The rather large number of sixty rules is the consequence of our aim to distribute the migration, thus revealing the distributed potential of the Paradigm-*McPal* tandem for system adaptation by giving freedom to *McPhil*s as delegates. As final remark we note, neither the STDs of *Phil*s and *Fork*s nor their roles *Eater*, *ForLH*, and *ForRH* roles had to be changed: the migration is fully situated within the coordination of the five ongoing collaborations. Again, *Phil* and *Fork* components remain running while the system migrates, dynamically indeed.

6 Discussion and concluding remarks

In the setting of component-based system development, we have addressed dynamic system adaptation without any form of quiescence. By using the coordination modeling language Paradigm, in combination with the special component *McPal*, we particularly underlined the suitability of the approach for dynamic adaptation in a distributed manner. The distributed potential of the Paradigm-*McPal* tandem is our main result, actually revealed through delegation among helpers. Concrete form to the distributive aspect is given via the dining philosophers example: letting the system adapt itself from a rather bad solution (deadlock) to a substantially better one having neither deadlock nor starvation.

In the context of the example, the distributed character of the adaption produces another three new results as spin-off, all three showing a wider reach

of the approach: (*i*) creation/deletion of STDs, (*ii*) adaptation with self-healing, (*iii*) behaviour computation. We elaborate on the three of them first.

In line with the coordination features offered by Paradigm, distribution of adaptation is achieved through delegation. Moreover, as adaptation is towards an originally unforeseen to-be solution, delegation thereof is brought into action by *McPal*. This results in concrete delegation to originally unforeseen components $McPhil_i$, one per collaboration $Phil2Forks_i$. As the *McPhil* components exist neither at the time the as-is solution is ongoing with *McPal* in hibernation nor at the time the to-be solution is ongoing with *McPal* in hibernation, in this case we model both STD creation and STD deletion in Paradigm, at the start and at the end of *McPal*'s non-hibernating phase *Migrating*, respectively. Modeling creation and deletion is achieved by simulating it via the phases of the various $McPhil_i(Evol)$ roles: creation of $McPhil_i$ when bringing it to life by leaving phase *Passive*; deletion of $McPhil_i$ when taking its life by returning to phase *Passive*. This way, STDs for components and for their roles can easily be created and deleted in a dynamically consistent manner, as all this comes down to suitable coordination.

As explained at the start of Section 4, coordinating adaptation, referred to as migration, is being modeled in state *JITting* such that different to-be situations can be reached, possibly through different migration trajectories. Accordingly, the migration model distributes the migration coordination among five helper $McPhil_i$, with the initial aim of locally achieving a reasonable result. Then *McPal*, by centrally collecting the partial results and comparing them in state *Delegated*, redistributes additional, specific alignment directives among the same five helper $McPhil_i$. After execution of the directives, final results are gathered and compiled into the particular to-be solution arising from the distributed migration coordination effort. The self-healing aspect, explicitly present in this example, lies in the activities occurring in state *Delegated* in view of selecting one out of eight outgoing action-transitions to state *Gathering*: rules (25)–(32) specify which particular alignment has to be done. The selection decision is the self-healing: it is solely based on trap information, certain combinations of five *halfwayL* vs. *halfwayR* traps having been entered. This means, it is solely based on intermediate migration results. Only in case of the two actions *goAheadLR* or *goAheadRL* the self-healing is empty; in the six other cases there is at least one adjustment from *L*-order to *R*-order or vice versa, and often two. Please note, such adjustments indeed arise on-the-fly of the still ongoing migration. Also interesting to note is, the self-healing directives are given at the level of *McPal*, the self-healing directives are performed at the (lower) delegation level of the five helper $McPhil_i$, very much in line with the architectural ideas in [15].

The above form of self-healing is finalized in *McPal*'s state *Gathering*. There the final to-be model is compiled into Crs_{toBe}, through composition of smaller model fragments composed to that aim by each helper $McPhil_i$. Fragments are about behaviour, so their composition certainly is behaviour computation, at the level of *McPal* as well as at the level of each $McPhil_i$. Thus, our behaviour computation is a distributed computation.

Another interesting feature of the example is, the seamless zipping of a conductor into a choreography, turning it into an 'equivalent' orchestration. Conversely, the seamless zipping of a conductor out of an orchestration, turns it into the 'equivalent' choreography. In this perspective, the temporary conductor $McPhil_i$ is reminiscent to the notion of a 'scaffold' in [20]. In our example, through the additional *Evol* role of a conductor $McPhil_i$, the scaffold has additional flexibility, changing phase-wise, while the model remains ongoing during alterations as usual.

As one might have observed, quite some redundancy appears in the above. (*i*) Paradigm has it in the role concept, repeating essence of component dynamics in view of *exogenous coordination* via consistency rules. (*ii*) Two roles per *Fork* introduce even more redundancy in view of architectural separation of five collaborative concerns. Behavioural redundancy is present too, organized in line with the five collaboarations $Phil2Forks_i$: (*iii*) After any helper $McPhil_i$ has communicated its *partial result*, it possibly has to undo the partial result. Or (*iv*) $McPhil_i$ possibly does essentially nothing, as partial result and local as-is as well as local to-be collaboration remain unchanged (L: left fork first, as always). This means, within the *environment* of the other four ongoing collaborations, a single $McPhil_i$'s behaviour computation *robustly* meanders towards its final result instead of going there straightforwardly.

During the final panel session at FACS 2010 the above four italicized characteristics –robust instead of correct, environment as first class citizen, exogenous coordination, partial results– have been positioned [7] as crucial for service-orientation in comparison to component technology. They reflect the additional flexibility service-orientation has to offer, when taking the next step from component technology. In Paradigm, these characteristics arise from redundancy designed on purpose: in language, in model structure and in model dynamics.

Recently, the Paradigm-*McPal* tandem is being deployed within Edafmis. The ITEA-project Edafmis aims at innovative integration of ICT-support from different advanced imaging systems into non-standard medical intervention practice, such that all flexibility needed during such interventions can be sustained smoothly and quickly, adequately and pleasantly. Particularly, the possibility for distributed migrations, as presented here, is of great value.

As presenting our model uses the full size of the paper, we are not able to address formal verification and further analysis of the migration here. We do have some results already. In future work we will report on it in more detail, in combination with other interesting migrations of dining philosophers.

References

1. M. Alia et al. Managing distributed adaptation of mobile applications. In J. Indulska and K. Raymond, editors, *Proc. DAIS 2007*, pages 104–118. LNCS 4531, 2007.
2. R. Allen, R. Douence, and D. Garlan. Specifying and analyzing dynamic software architectures. In E. Astesiano, editor, *Proc. FASE 1998*, pages 21–37. LNCS 1382, 1998.

3. S. Andova, L.P.J. Groenewegen, J. Stafleu, and E.P. de Vink. Formalizing adaptation on-the-fly. In G. Salaün and M. Sirjani, editors, *Proc. FOCLASA 2009*, pages 23–44. ENTCS 255, 2009.
4. S. Andova, L.P.J. Groenewegen, J.H.S. Verschuren, and E.P. de Vink. Architecting security with Paradigm. In R. de Lemos et al., editor, *Architecting Dependable Systems VI*, pages 255–283. LNCS 5835, 2009.
5. S. Andova, L.P.J. Groenewegen, and E.P. de Vink. Dynamic consistency in process algebra: From Paradigm to ACP. *Science of Computer Programming*, 2010. `doi:10.1016/j.scico.2010.04.011`, 45pp.
6. S. Andova, L.P.J. Groenewegen, and E.P. de Vink. Towards dynamic adaptation of probabilistic systems. In *Proc. ISoLa 2010, Part II, Heraclion*. LNCS 6416, 2010. To appear.
7. F. Arbab, 2010. Personal communication.
8. N. Bencomo et al. Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems. In *Proc. DSPL 2008*, pages 23–32. Limerick, 2008.
9. K.N. Biyani and S.S. Kulkarni. Assurance of dynamic adaptation in distributed systems. *Journal of Parallel Distributed Computing*, 68:1097–1112, 2008.
10. J.S. Bradbury et al. A survey of self-management in dynamic software architecture specifications. In D. Garlan, J. Kramer, and A.L. Wolf, editors, *Proc. WOSS 2004*, pages 28–33. ACM, 2004.
11. A. Bucchiarone et al. Self-repairing systems modeling and verification using agg. In *Proc. WICSA/ECSA 2009, Cambridge*, pages 181–190. IEEE, 2009.
12. H. Ehrig et al. Formal analysis and verification of self-healing systems. In D. Rosenblum and G. Taentzer, editors, *Proc. FASE 2010*, pages 139–155. LNCS 6013, 2010.
13. L. Groenewegen and E. de Vink. Evolution-on-the-fly with Paradigm. In P. Ciancarini and H. Wiklicky, editors, *Proc. Coordination 2006*, pages 97–112. LNCS 4038, 2006.
14. J. Kramer and J. Magee. The evolving philosophers problem: dynamic change management. *IEEE Transactions on Software Engineering*, 16:1293–1306, 1990.
15. J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In L.C. Briand and A.L. Wolf, editors, *Proc. FOSE 2007*, pages 259–268. IEEE, 2007.
16. J. Magee and J. Kramer. Dynamic structure in software architectures. *SIGSOFT Software Engineering Notes*, 21:3–14, 1996.
17. T. Melliti, P. Poizat, and S.B. Mokhtar. Distributed behavioural adaptation for the automatic composition of semantic services. In J.L. Fiadeiro and P. Inverardi, editors, *Proc. FASE 2008*, pages 146–162. LNCS 4961, 2008.
18. B. Morin et al. An aspect-oriented and model-driven approach for managing dynamic variability. In K. Czarnecki et al., editor, *Proc. MoDELS 2008*, pages 782–796. LNCS 5301, 2008.
19. M.-T. Segarra and F. André. A distributed dynamic adaptation model for component-based applications. In I. Awan et al., editor, *Proc. AINA 2009*, pages 525–529. IEEE, 2009.
20. A.W. Stam. *Interaction Protocols in PARADIGM*. PhD thesis, LIACS, Leiden University, 2009.
21. M. Yarvis, P. Reiher, and G.J. Popek. Conductor: A framework for distributed adaptation. In *Proc. HOTOS 1999, Rio Rico*, pages 44–51. IEEE, 1999.
22. J. Zhang, H.J. Goldsby, and B.H.C. Cheng. Modular verification of dynamically adaptive systems. In K.J. Sullivan et al., editor, *Proc. AOSD 2009*, pages 161–172. ACM, 2009.