Towards Dynamic Adaptation of Probabilistic Systems

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

[1] Formal Methods, TU Eindhoven, The Netherlands
[2] FaST Group, LIACS, Leiden University, The Netherlands

Abstract. Dynamic system adaptation is modeled in the coordination language Paradigm as coordination of collaborating components. A special component McPal allows for addition of new behavior, of new constraints and of new control in view of a new collaboration. McPal gradually adapts the system dynamics. It is shown that the approach also applies to the probabilistic setting. For a client-server example, where McPal adds, step-by-step, probabilistic behavior to deterministic components, precise modeling of changing system dynamics is achieved. This modeling of the transient behavior, spanning the complete migration range from as-is collaboration to to-be collaboration, serves as a stepping stone to quantitative analysis of the system during adaptation.

1 Introduction

Many systems today are affected while running by changes in their operational environment, while they cannot be shutdown to be updated and restarted again. Instead, dynamic adaptive systems must be able to manage adaptation steps on-the-fly to accommodate a new plan. Dynamic adaptive systems usually consist of interactive components, architecturally organized. However, system adaptation requires proper coordination. A carefully chosen coordination should guarantee that, during the adaptation, the system functionality is neither interrupted nor disturbed, and non-functional quantities, though possibly changing, should not exceed allowed bounds.

The coordination language Paradigm has been shown suitable to model dynamic adaptation [11, 2] without the need of quiescence, i.e. no component has to be isolated from the system before being changed. In Paradigm, a system architecture is organized along specific collaboration dimensions, called partitions. A partition of a component specifies various phases the component goes through when a protocol is executed. Phases are temporarily valid constraints on ongoing component behaviour. In the protocol, at a higher layer in the architecture, the component participates via its role, an abstract representation of the phases. A protocol coordinates the phase transfers for the components involved. Progress within a phase is completely local to the component. In fact, the use of phase transfer instead of state transfer, where phases allow components to pursue their local dynamics, is the key concept of Paradigm. This makes it possible to model

architectural changes and, at the same time, to model behavioural changes per component and per collaboration. In [4] an encoding of Paradigm models for the mCRL2 model checker is presented. The connection is exploited in [2]: (*i*) to verify that the system under adaptation indeed migrates from original to new behaviour; (*ii*) to perform a qualitative analysis of the adaptation itself. Such a formal analysis is relevant for detecting conflicts and revealing inconsistencies, in particular in case of multiple, simultaneous adaptation, guided by different change managers.

In this paper we extend the approach of modeling system adaptation with Paradigm and subsequent model checking of transitional properties and invariants by considering Markov decision processes (MDPs) instead of state-transition diagrams (STDs). We revisit our example of a critical section problem with four clients and one server. Thus, following Paradigm's methodology, the source and target behaviours are modeled as collaborative MDPs, which are STDs in the degenerate case. An adaptation strategy is given as well. Starting from deterministic round-robin servicing we aim to evolve to a client-to-serve selection based on a probability distribution. By guiding components phase by phase, probabilistic behavior is added gradually to deterministic components. They smoothly migrate from the source model to the target model. As mCRL2 does not allow probabilities, we now encode the whole adaptation trajectory in Prism [12]. The Prism specification consists of several modules, one for each client component and one for the server component. Within each module, both the detailed behaviour of the component is captured as well as the more global phase constraints and transfers. In fact, dynamic constraints, essential in Paradigm for the coordination of collaboration, can straightforwardly be specified with Prism via guards. So-called consistency rules that enforce multi-party synchronization in Paradigm at the level of phases, are distributively encoded by reactive commands sharing a label. Here, the specification language of the model checker fits hand in glove with the component interaction mechanism of Paradigm. We were able to generate a complete model of the dynamic adaptation, on which further qualitative and quantitative analysis on the transitional behaviour of the system is conducted. For instance, it is possible to guarantee mutual exclusion during adaptation. As for the quantitative properties it is, for instance, possible to compute the expected time needed for the system to migrate and to calculate the maximal waiting time for service during migration.

Related work Most of the existing approaches, [8, 6, 9, 16] to mention a few of them, focus on adaptive software architectures, where functionalities, considered as black boxes, are connected via ports. Following [13], new behavior is introduced by replacing an existing component by a new version. However, a component can only be removed if it is quiet and all affected components are passive. Thus, the actual adaptation is mainly achieved by reorganizing the architecture. [1] and [15] are the first efforts to analyze system adaptation with model checking.

Though recognized as an important issue and challenge [14], formal analysis of transitional behaviour of dynamic adaptive systems has triggered attention

only recently. The approach of [18, 19] is closest to ours: Cheng et al. manage to model and formally analyse behavioural adaptation through weakening the need for quiescence. Their formal modeling uses Petri Nets and automata. In this way, functional properties expressed in an LTL-based logic, can be formally verified. A drawback is that different adaptation trajectories cannot be combined in a single model; also, adaptation is not being coordinated. In none of the approaches mentioned quantitative analysis is addressed.

To the best of our knowledge, none of the existing approaches supports modeling and property analysis of dynamic graceful adaptation without quiescence, neither qualitative nor quantitative. However, for simple value adaptation, as for instance in [17], service reconfiguration is addressed using quality description parameters to determine potential target configurations. The supporting verification framework includes a model checker, e.g., to verify reachability of configurations. The AADL modeling language of [7] supports dynamic reconfiguration of component connections. The language is rather expressive, allowing to specify timed, stochastic as well as hybrid systems. It is supported by a verification environment, including MRMC for model checking quantitative properties.

Organization of the paper Section 2 introduces Paradigm by example, discussing the central notions for the deterministic version of the client-server system. A probabilistic service policy is modeled in Section 3, the migration from round-robin to probabilistic service is covered in Section 4. The encoding in Prism and further analysis of the adaptation are discussed in Section 5. The last section wraps up with conclusions and future work.

2 As-Is Situation: Deterministic Round Robin Service

This section presents a first variant of a Client-Server system: one Server component and four Client components, with merely deterministic behaviour. The five components collaborate on the basis of a round robin scheme. We shall refer to this variant as *as-is*.

Coordination language Paradigm can specify coordination solutions for foreseen as-is collaborations [3, 5, 4], for originally unforeseen to-be collaborations as well as for migration, i.e. ongoing but smoothly changing collaboration during adaptation from as-is to to-be [2, 11]. To explain how, we first look briefly at Paradigm's coordination specification through the example of the Client-Server system. Second and briefly too, we introduce Paradigm's special component McPal, not influencing the system at all (as yet), but present in view of later system migration. The references give more technical background.

Paradigm has five basic notions: STD, phase, (connecting) trap, role and consistency rule. (Definitions are in [5, 3], semantics in [5].) Figure 1 visualizes four of the notions for the Clients of the as-is system. Component behaviour is specified by STDs, state-transition diagrams. Figure 1a gives the STD for each Client component, in UML style. It says, Client_i starts in state Out and has cyclic behaviour, forever visiting its five states by subsequently taking actions enter, choose, explain, thank, leave. The idea is, by being in state Busy, a Client

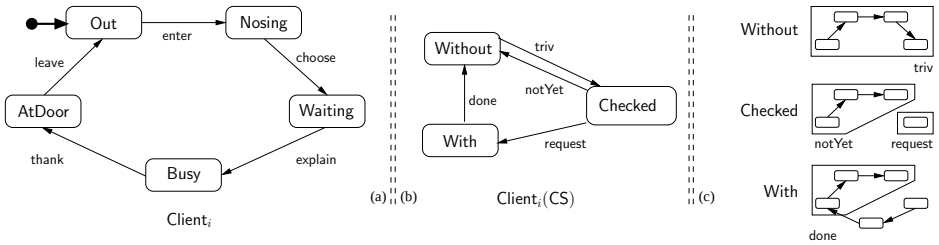

Fig. 1. (a) Client_i STDs, (b) their CS role dynamics by (c) phase/trap constraints

should exclusively occupy the one Server. In a nut-shell, this requirement lies at the basis of the critical section collaboration (CS) of the as-is system. The CS collaboration should provide suitable, simultaneous constraints on the Client behaviours, such that (i) never two or more Clients are in Busy at the same time; (ii) after arrival in Waiting, permission to visit Busy will be given sooner or later, as by being in Waiting a Client is asking for the permission.

Within Paradigm, a component participating in a collaboration does not contribute to the collaboration via its STD behaviour directly. Instead, the component contributes via a so-called *role*, being a different STD for the component, exhibited at a more global level. Figure 1b specifies role $\mathsf{Client}_i(\mathsf{CS})$ that Client_i contributes to the CS collaboration. States of role $\mathsf{Client}_i(\mathsf{CS})$ are so-called *phases* of the Client_i STD (Figure 1c): temporarily valid behavioural *constraints imposed* on the STD Client_i. Any current role state (phase) has as semantics: it keeps the behaviour of the STD it is a role of, *constrained to that phase*. Figure 1b and, correspondingly, Figure 1c mention three phases: Without, Checked, With. Here, Without prohibits a Client to be in Busy; With permits a Client to enter and to leave Busy once; Checked is an interrupted form of Without, to see whether a Client asks permission for entering Busy. In Figure 1c each phase is additionally decorated with one or more polygons, each polygon grouping states of that phase into a set. Polygons visualize so-called *traps*: a trap, as set of states, once entered, cannot be left as long as the phase remains the current role state. A trap serves as a guard for a phase transfer. Therefore, traps label transitions in a role STD, cf. Figure 1b. If all states in a trap serve as starting states of the next phase, the trap is called *connecting from* the one phase to the next.

Thus, role $\mathsf{Client}_i(\mathsf{CS})$ behaviour, Figure 1b, expresses possible sequences of phase transfers: the phase transfer from Without to Checked is generally possible, as trap triv is always connecting from Without to Checked; the step from Checked to With is only possible if connecting trap request has been entered, which means, if Client_i asks the permission in state Waiting; otherwise, via connecting trap notYet, the phase transfer is from Checked back to Without; the phase transfer from With to Without is only possible after connecting trap done has been entered.

The STD of the Server is visualized in Figure 2a. The idea of Server is, (i) being in state $\mathsf{Checking}_i$ means, Client_i behaviour is kept within phase Checked while the other Clients are being kept within phase Without; (ii) being in state $\mathsf{Helping}_i$ means, Client_i behaviour is kept within phase With while the other Clients are being kept within phase Without. Note Server's round robin strategy

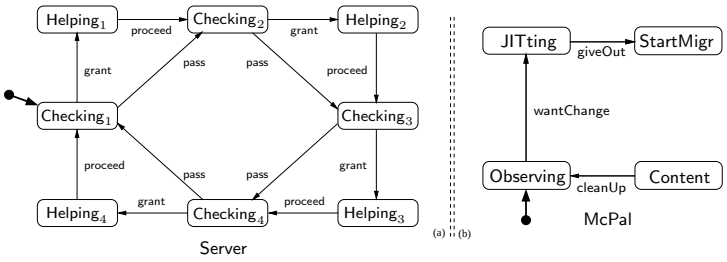

Fig. 2. STDs (a) Server and (b) McPal; the latter in Hibernating form only

in addressing the next Client_{i+1}, after having checked and possibly even helped Client_i.

In view of possible adaptation, the additional STD McPal, acting as an adaptation change conductor, is in place in its so-called hibernating form, visualized in Figure 2b. The idea is, as long as adaptation is not triggered, McPal is as yet interfering neither with Clients and Server nor with their collaboration. In particular, from Figure 2b we see McPal, starting in Observing, can go as far as StartMigr. But without interfering with itself, it can neither reach state Content nor return to state Observing. So the next idea is, once McPal has reached StartMigr it still has not started whatever adaptation of the as-is system, but by then it just-in-time has prepared such later migration through taking its last step giveOut, thus updating the specification of the original Paradigm model; to that aim the model specification is stored in McPal's local variable Crs. This is *reflectivity* of Paradigm models: a model contains its own specification and it can update it.

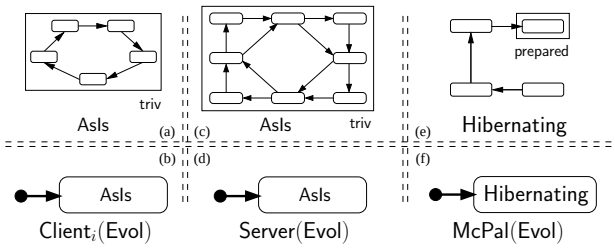

Fig. 3. Evol phases and roles: Client_i, Server, McPal

Moreover, in view of whatever possible change McPal might wish to exert on the STDs Server, Clients and McPal itself, each such STD has an Evol role that does not restrict their dynamics at all, as yet, see Figure 3. Note that each Evol role comprises exactly one state (and no steps).

To formulate a coordination solution for a collaboration in terms of constraints specified earlier, single steps from different roles, are synchronized into one protocol step. A protocol step can be coupled with one detailed step of a so-called conductor component. Also, variables local to the conductor can be updated. It is through a *consistency rule*, Paradigm specifies a protocol step: (*i*) at the left-hand side of an asterisk $*$ the one conductor step is given, if relevant; (*ii*) the right-hand side lists the role steps being synchronized; (*iii*) optionally, a change clause can be given for updating variables, in particular the variable Crs

containing the full model specification including the consistency rules, cf. [11]. A consistency rule with a conductor step is called an *orchestration* step, a consistency rule without it is called a *choreography* step.

The consistency rules for the orchestration of the $\mathsf{Client}_i(\mathsf{CS})$ roles, conducted by Server, are given by the first three rules below. Rule (1) says, if STD Server is in $\mathsf{Checking}_i$ and if role $\mathsf{Client}_i(\mathsf{CS})$ is in Checked and trap request of Checked has been entered, then Server can take step grant, thereby enforcing $\mathsf{Client}_i(\mathsf{CS})$ to take step request synchronously. Similarly, rule (2) synchronously couples Server's step proceed from $\mathsf{Checking}_i$ to $\mathsf{Checking}_{i+1}$ with $\mathsf{Client}_i(\mathsf{CS})$'s role step notYet from Checked to Without as well as with $\mathsf{Client}_{i+1}(\mathsf{CS})$'s role step triv from Without to Checked. Etc.

$$\mathsf{Server}\colon \mathsf{Checking}_i \xrightarrow{grant} \mathsf{Helping}_i \ * \ \mathsf{Client}_i(\mathsf{CS})\colon \mathsf{Checked} \xrightarrow{request} \mathsf{With} \tag{1}$$

$$\mathsf{Server}\colon \mathsf{Checking}_i \xrightarrow{pass} \mathsf{Checking}_{i+1} \ * \quad \mathsf{Client}_i(\mathsf{CS})\colon \mathsf{Checked} \xrightarrow{notYet} \mathsf{Without},\ \mathsf{Client}_{i+1}(\mathsf{CS})\colon \mathsf{Without} \xrightarrow{triv} \mathsf{Checked} \tag{2}$$

$$\mathsf{Server}\colon \mathsf{Helping}_i \xrightarrow{proceed} \mathsf{Checking}_{i+1} \ * \quad \mathsf{Client}_i(\mathsf{CS})\colon \mathsf{With} \xrightarrow{done} \mathsf{Without},\ \mathsf{Client}_{i+1}(\mathsf{CS})\colon \mathsf{Without} \xrightarrow{triv} \mathsf{Checked} \tag{3}$$

$$\mathsf{McPal}\colon \mathsf{JITting} \xrightarrow{giveOut} \mathsf{StartMigr} \ * \ \mathsf{McPal}\colon [\,\mathsf{Crs} := \mathsf{Crs} + \mathsf{Crs}_{\mathsf{migr}} + \mathsf{Crs}_{\mathsf{toBe}}\,] \tag{4}$$

$$\mathsf{McPal}\colon \mathsf{Content} \xrightarrow{cleanUp} \mathsf{Observing} \ * \ \mathsf{McPal}\colon [\,\mathsf{Crs} := \mathsf{Crs}_{\mathsf{toBe}}\,] \tag{5}$$

Please note, McPal is not involved in any coordination at all, rules (4) and (5), as no role steps are coupled with its non-role steps. But it prepares such migration coordination when going to state StartMigr (4): by extending the specification of the as-is coordination with the coordination for the adaptation as well as for the to-be situation. The specification's extension is being compensated (later) by McPal's step cleanUp when returning to Observing (5): by reducing the coordination specification to the to-be situation only.

3 To-Be Situation: Stationary Probabilistic Service

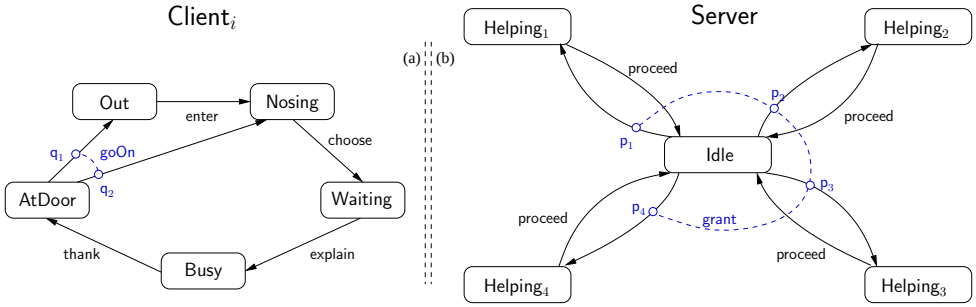

Fig. 4. STDs (a) Client_i to-be, (b) Server to-be

In this section we indicate, by example only, how Paradigm models can be endowed with probabilistic transitions. Thus, our STDs become Markov decision processes. For the probabilistic example we again consider a collaboration between four Client STDs and a Server STD. Moreover, in view of the adaption we

want to discuss later, these probabilistic STDs will figure as to-be versions of the original non-probabilistic STD versions discussed in Section 2.

The STD of the Client_i in its new to-be form, see Figure 4a, has a new action goOn, replacing action leave. As one can see, the probabilistic transition labeled with action goOn points to two target states, to state Out, with probability q_1 and to state Nosing with probability q_2. Note $\mathsf{q}_1, \mathsf{q}_2 \geq 0$. As there are no other target states, we have $\mathsf{q}_1 + \mathsf{q}_2 = 1$. Graphically, the two arrows, leaving the same state and referred to by the same action, are *shackled* by a (blue) dashed line.

The to-be form of process Server has changed rather more drastically, see Figure 4b. In the new state Idle there is one action grant available with four probabilistic outcomes: the four arrows are from Idle to the four states $\mathsf{Helping}_i$ and have probabilities $\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4 \geq 0$, respectively, $\mathsf{p}_1 + \mathsf{p}_2 + \mathsf{p}_3 + \mathsf{p}_4 = 1$. The idea of Server is, in Idle it serves no Client at all, and in $\mathsf{Helping}_i$ it serves Client_i exclusively. As the probability to go to $\mathsf{Helping}_i$ is always the same, although possible different fo each i, this kind of service strategy is called a stationary probabilistic service.

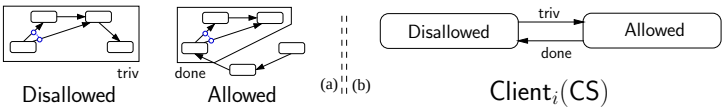

Fig. 5. Client_i to-be: (a) phases and traps, for (b) role $\mathsf{Client}_i(\mathsf{CS})$

The to-be role $\mathsf{Client}_i(\mathsf{CS})$ as given in Figure 5 differs from the as-is role in Figure 1 in two points. (*i*) The new versions of phases Without and With are called Disallowed and Allowed respectively, as in the to-be situation they have to contain action goOn instead of leave. (*ii*) For phase Checked there is no to-be version, as granting service to any Client_i is done independently from Client_i asking for it. As a consequence, the service turn should terminate immediately if Client_i doesn't need it right then. And indeed it does, as trap done of phase Allowed is entered exactly when phase Allowed gets imposed: process Client_i is in one of the states of trap done *already* because it did *not yet* request for the service turn. Please note, this is mimicked in the Server behaviour by the probabilistic transition grant leading from Idle immediately to state $\mathsf{Helping}_i$.

The consistency rules for the orchestration of the to-be $\mathsf{Client}_i(\mathsf{CS})$ roles, conducted by the to-be Server, are as follows.

$$\begin{aligned}
&\mathsf{p}_1 \cdot [\,\mathsf{Server}\colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Helping}_1 \;*\; \mathsf{Client}_1(\mathsf{CS})\colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Allowed}\,] \oplus \\
&\mathsf{p}_2 \cdot [\,\mathsf{Server}\colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Helping}_2 \;*\; \mathsf{Client}_2(\mathsf{CS})\colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Allowed}\,] \oplus \\
&\mathsf{p}_3 \cdot [\,\mathsf{Server}\colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Helping}_3 \;*\; \mathsf{Client}_3(\mathsf{CS})\colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Allowed}\,] \oplus \\
&\mathsf{p}_4 \cdot [\,\mathsf{Server}\colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Helping}_4 \;*\; \mathsf{Client}_4(\mathsf{CS})\colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Allowed}\,]
\end{aligned} \tag{6}$$

$$\mathsf{Server}\colon \mathsf{Helping}_i \xrightarrow{proceed} \mathsf{Idle} \;*\; \mathsf{Client}_i(\mathsf{CS})\colon \mathsf{Allowed} \xrightarrow{done} \mathsf{Disallowed} \tag{7}$$

$$\mathsf{McPal}\colon \mathsf{JITting} \xrightarrow{giveOut} \mathsf{StartMigr} \;*\; \mathsf{McPal}\colon [\,\mathsf{Crs} := \mathsf{Crs} + \mathsf{Crs}_{\mathsf{migr}} + \mathsf{Crs}_{\mathsf{toBe}}\,] \tag{8}$$

$$\mathsf{McPal}\colon \mathsf{Content} \xrightarrow{cleanUp} \mathsf{Observing} \;*\; \mathsf{McPal}\colon [\,\mathsf{Crs} := \mathsf{Crs}_{\mathsf{toBe}}\,] \tag{9}$$

Please note, for the coordination, we have coupled Server action grant in state

Idle, via its probabilistic outcomes in terms of four possible detailed steps, to steps in four different roles (6). But each of these role steps is deterministically coupled to one specific detailed Server step. In this manner, the conductor throws the dice and the four participants, each one at the level of its CS role, deterministically obey to the probabilistic outcome. Moreover note, here too McPal is not involved in any coordination at all, as it only gets involved when the ongoing orchestration is to be adapted (8), (9).

4 From Deterministic to Probabilistic Service

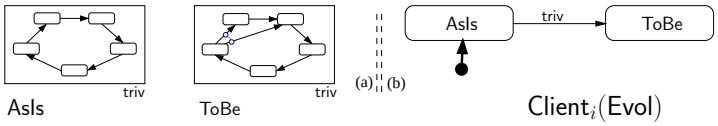

Fig. 6. Client$_i$ during migration: (a) phases and traps, for (b) role Client$_i$(Evol)

Based on the as-is and to-be versions of process Client$_i$ in Sections 2 and 3 we observe the following: (i) Phase ToBe from Figure 6a exactly specifies the constraint on Client$_i$ needed for its Evol role during the to-be situation, as it does not prohibit any detailed step that should be able to occur; (ii) Figure 6b then specifies a feasible migration in terms of role Client$_i$(Evol), in one go from AsIs to ToBe. Here we do not need any migration phase in between.

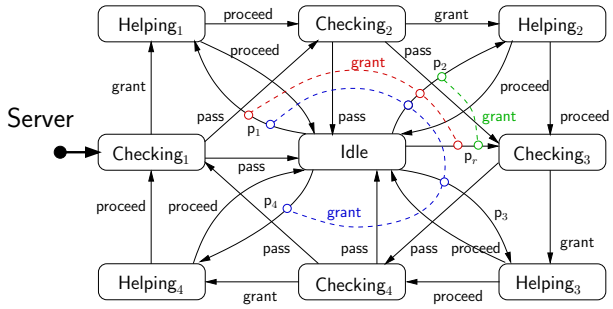

Fig. 7. STD Server during migration.

In preparation of role Server(Evol), Figure 7 gives the detailed steps of Server during as-is, during to-be as well as during adaptation from as-is to to-be. Based on the foregoing Sections 2 and 3, it is easy to recognize in Figure 7, the steps from both the as-is and the to-be situations. But the other steps, apparently present in view of the intermediate migration trajectories, are not so clear.

In particular, the four probabilities $\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4$ are those from Section 3. But probability p_r, labeling the transition from Idle to Checking$_3$, serves two purposes and, accordingly, has two values: (i) $\mathsf{p}_r = \mathsf{p}_{134} = \mathsf{p}_1 + \mathsf{p}_3 + \mathsf{p}_4$, if linked to only the transition from Idle to Helping$_2$ (2-legs-shackle); (ii) $\mathsf{p}_r = \mathsf{p}_{34} = \mathsf{p}_3 + \mathsf{p}_4$, if linked to only the two transitions from Idle to Helping$_2$ and to Helping$_1$ (3-legs-shackle). Thus there are three shackles, linking either two transitions leaving state Idle, or

three or four such transitions –the last one is the 4-legs-shackle from Section 3. To enlarge the entanglement even more, each shackle has the same label grant, thus referring to three different actions of that name, available in state Idle for making a transition from it; two of these must be there in view of the adaptation, as they neither belong to the as-is nor to the to-be situation.

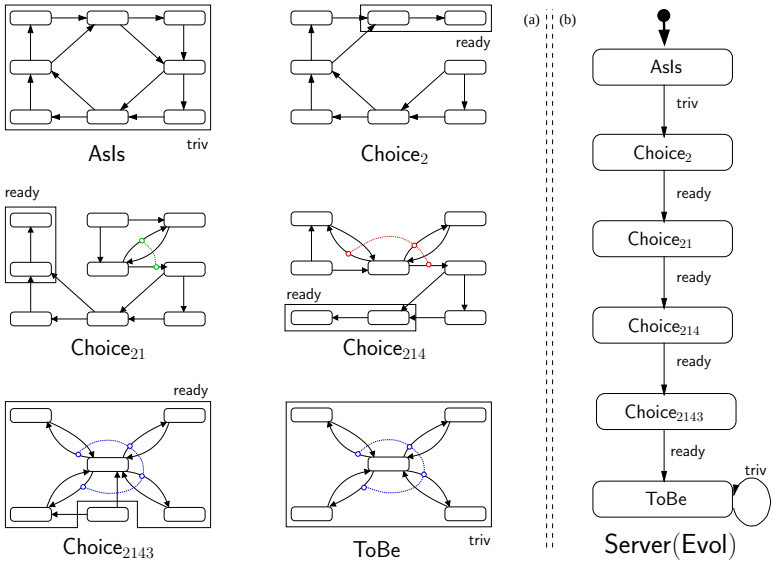

Fig. 8. Server during migration: (a) phases and traps, for (b) role Server(Evol)

Figure 8 repairs the unclarity, by giving the historical overview of the detailed dynamics of the Server through the six phases of partition Evol: phase AsIs visualizing the original, deterministic service provision, phase ToBe visualizing the target, probabilistic service provision and the actual migration phases Choice_2, Choice_{21}, Choice_{214} and Choice_{2143} visualizing in that order, how to get rather gradually from as-is to to-be service provision.

In more detail one can see the following: (*i*) In Choice_2 the deterministic round robin approach can only go as far as addressing Client_2. (*ii*) Upon addressing Client_2, it is decided for the last time in round robin fashion, whether it gets the turn. From then on, once Server within phase Choice_{21} returns to state Idle, process Client_2 has probability p_2 to get the service turn right then. But the other three Clients, with probability p_{134}, will be served instead in the usual round robin fashion. So within phase Choice_{21} we have action grant labeling the 2-legs-shackle. (*iii*) Next, as soon as Client_1 is addressed, it is decided for the last time in round robin fashion whether it gets the turn. From then on, once Server within phase Choice_{214} returns to state Idle, process Client_1 has probability p_1 to get the service turn right then. But the other two Clients, with probability p_{34}, will be served instead in the usual round robin fashion. So within phase Choice_{214} we have action grant labeling the 3-legs-shackle. (*iv*) Then, as soon as Client_4 is addressed, it is decided for the last time in round robin fashion whether it gets

the turn. From then on, once Server within phase Choice_{2143} returns to state Idle, process Client_4 has probability p_4 to get the service turn right then. And from then on too, the last Client_3 will be served with probability p_3. So within phase Choice_{2143} we have action grant labeling the 4-legs-shackle, as needed in the to-be situation. (v) Finally, as soon as Client_3 is addressed, the to-be situation is indeed considered as reached and the migration coordination will stop accordingly.

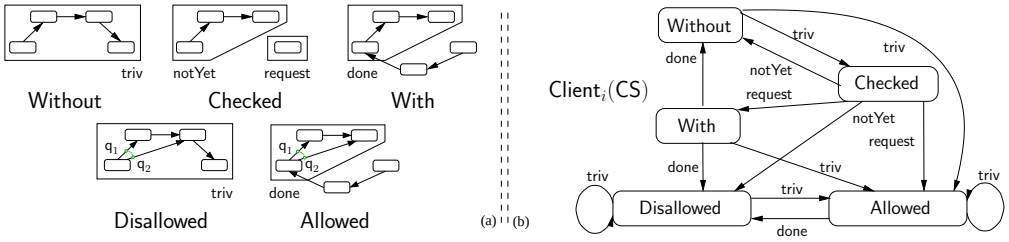

Fig. 9. $\mathsf{Client}_i(\mathsf{CS})$ during migration: (a) phases and traps, for (b) role $\mathsf{Client}_i(\mathsf{CS})$

The adaptation of CS role of the Clients from the as-is dynamics in Figure 1b to the to-be dynamics in Figure 5 is visualized in Figure 9.

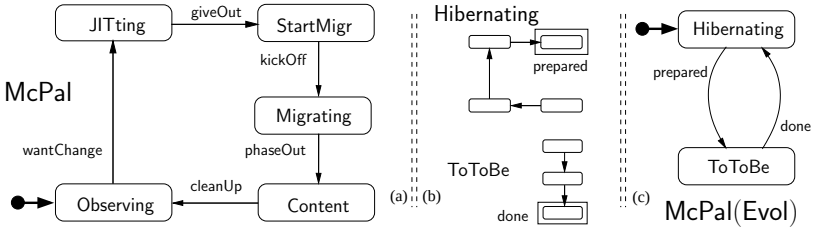

Fig. 10. Full McPal: (a) STD, (b) phases and traps, (c) role McPal(Evol)

The full McPal STD during adaptation from as-is to to-be situation is drawn in Figure 10, together with its Evol role and phases and traps for it. Please note, McPal in its phase ToToBe has only one state Migrating in between its states StartMigr and Content. The step to Migrating is meant to start the migration by conducting synchronous phase transfers in the Evol roles of Server as well as of all Clients. By doing so, McPal delegates the coordination of the remaining steps of the Server(Evol) role to either the role itself (choreography) or to Server (orchestration). Hence McPal has to wait in Migrating until Server actually achieves the coordination task just delegated to it. Finally, the step from Migrating coincides with McPal observing that Server has finished the task delegated.

The consistency rules specifying how to coordinate the adaptation along the lines visually clarified above, are given below. They appear in four groups. Please note, rules (10)-(31), together with the corresponding STDs constitute the value of McPal's local variable $\mathsf{Crs}_{\mathsf{migr}}$. Likewise, the value of McPal's local variable $\mathsf{Crs}_{\mathsf{toBe}}$ contains the rules (6)-(9) from Section 3, while Crs contains the rules (1)-(5) and corresponding specifications from Section 2 as its initial value.

The first group of rules given here are for McPal. In particular they cover the two choreography steps from phase Hibernating to ToToBe and from ToToBe

back to Hibernating. Moreover, they cover the coordination conducted by McPal when within phase ToToBe. Please note, the delegation of adaptation tasks from McPal to Server is captured in the rule (11).

$$* \ \mathsf{McPal(Evol)} \colon \mathsf{Hibernating} \xrightarrow{prepared} \mathsf{ToToBe} \tag{10}$$

$$\begin{aligned} \mathsf{McPal} \colon \mathsf{StartMigr} \xrightarrow{kickOff} \mathsf{Migrating} \ * \ \mathsf{Server(Evol)} \colon \mathsf{AsIs} \xrightarrow{triv} \mathsf{Choice}_2, \\ \mathsf{Client}_1\mathsf{(Evol)} \colon \mathsf{AsIs} \xrightarrow{triv} \mathsf{ToBe}, \ \mathsf{Client}_2\mathsf{(Evol)} \colon \mathsf{AsIs} \xrightarrow{triv} \mathsf{ToBe}, \\ \mathsf{Client}_3\mathsf{(Evol)} \colon \mathsf{AsIs} \xrightarrow{triv} \mathsf{ToBe}, \ \mathsf{Client}_4\mathsf{(Evol)} \colon \mathsf{AsIs} \xrightarrow{triv} \mathsf{ToBe} \end{aligned} \tag{11}$$

$$\mathsf{McPal} \colon \mathsf{Migrating} \xrightarrow{phaseOut} \mathsf{Content} \ * \ \mathsf{Server(Evol)} \colon \mathsf{ToBe} \xrightarrow{triv} \mathsf{ToBe} \tag{12}$$

$$* \ \mathsf{McPal(Evol)} \colon \mathsf{ToToBe} \xrightarrow{migrDone} \mathsf{Hibernating} \tag{13}$$

All further rules are for Server, which guides the adaptation changes. As there are quite many of them, we split them into similar groups, corresponding to the remaining role steps of Server(Evol). First those guiding the transfer from Choice_2 to Choice_{21}.

$$* \ \mathsf{Server(Evol)} \colon \mathsf{Choice}_2 \xrightarrow{ready} \mathsf{Choice}_{21}, \ \mathsf{Client}_2\mathsf{(CS)} \colon \mathsf{Checked} \xrightarrow{notYet} \mathsf{Disallowed} \tag{14}$$

$$* \ \mathsf{Server(Evol)} \colon \mathsf{Choice}_2 \xrightarrow{ready} \mathsf{Choice}_{21}, \ \mathsf{Client}_2\mathsf{(CS)} \colon \mathsf{With} \xrightarrow{triv} \mathsf{Allowed} \tag{15}$$

$$* \ \mathsf{Server(Evol)} \colon \mathsf{Choice}_2 \xrightarrow{ready} \mathsf{Choice}_{21}, \ \mathsf{Client}_2\mathsf{(CS)} \colon \mathsf{Checked} \xrightarrow{request} \mathsf{Allowed} \tag{16}$$

$$\mathsf{Server} \colon \mathsf{Checking}_2 \xrightarrow{pass} \mathsf{Idle} \ * \ \mathsf{Client}_2\mathsf{(CS)} \colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Disallowed} \tag{17}$$

$$\mathsf{Server} \colon \mathsf{Checking}_2 \xrightarrow{grant} \mathsf{Helping}_2 \ * \ \mathsf{Client}_2\mathsf{(CS)} \colon \mathsf{Allowed} \xrightarrow{triv} \mathsf{Allowed} \tag{18}$$

$$\begin{aligned} \mathsf{p}_2 \cdot [\, \mathsf{Server} \colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Helping}_2 \ * \ \mathsf{Client}_2\mathsf{(CS)} \colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Allowed} \,] \ \oplus \\ \mathsf{p}_{134} \cdot [\, \mathsf{Server} \colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Checking}_3 \ * \ \mathsf{Client}_3\mathsf{(CS)} \colon \mathsf{Without} \xrightarrow{triv} \mathsf{Checked} \,] \end{aligned} \tag{19}$$

The first three choreography steps address the actual phase transfer from Choice_2 to Choice_{21} once trap ready of Choice_2 has been entered. The choreography is moreover coupled to relevant CS role steps of Client_2: phase transfers from Checked or With to Disallowed or Allowed, thus only now enabling the to-be probabilistic behaviour to Client_2 exclusively. The last three orchestration steps cover the deviating part of the CS protocol under migration during phase Choice_{21} only.

$$* \ \mathsf{Server(Evol)} \colon \mathsf{Choice}_{21} \xrightarrow{ready} \mathsf{Choice}_{214}, \ \mathsf{Client}_1\mathsf{(CS)} \colon \mathsf{Checked} \xrightarrow{notYet} \mathsf{Disallowed} \tag{20}$$

$$* \ \mathsf{Server(Evol)} \colon \mathsf{Choice}_{21} \xrightarrow{ready} \mathsf{Choice}_{214}, \ \mathsf{Client}_1\mathsf{(CS)} \colon \mathsf{With} \xrightarrow{triv} \mathsf{Allowed} \tag{21}$$

$$* \ \mathsf{Server(Evol)} \colon \mathsf{Choice}_{21} \xrightarrow{ready} \mathsf{Choice}_{214}, \ \mathsf{Client}_1\mathsf{(CS)} \colon \mathsf{Checked} \xrightarrow{request} \mathsf{Allowed} \tag{22}$$

$$\mathsf{Server} \colon \mathsf{Checking}_1 \xrightarrow{pass} \mathsf{Idle} \ * \ \mathsf{Client}_1\mathsf{(CS)} \colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Disallowed} \tag{23}$$

$$\mathsf{Server} \colon \mathsf{Checking}_1 \xrightarrow{grant} \mathsf{Helping}_1 \ * \ \mathsf{Client}_1\mathsf{(CS)} \colon \mathsf{Allowed} \xrightarrow{triv} \mathsf{Allowed} \tag{24}$$

$$\begin{aligned} \mathsf{p}_1 \cdot [\, \mathsf{Server} \colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Helping}_1 \ * \ \mathsf{Client}_1\mathsf{(CS)} \colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Allowed} \,] \ \oplus \\ \mathsf{p}_2 \cdot [\, \mathsf{Server} \colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Helping}_2 \ * \ \mathsf{Client}_2\mathsf{(CS)} \colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Allowed} \,] \ \oplus \\ \mathsf{p}_{34} \cdot [\, \mathsf{Server} \colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Checking}_3 \ * \ \mathsf{Client}_3\mathsf{(CS)} \colon \mathsf{Without} \xrightarrow{triv} \mathsf{Checked} \,] \end{aligned} \tag{25}$$

Finally, the consistency rules for transfer from Choice_{214} to Choice_{2143}.

$$* \ \mathsf{Server(Evol)} \colon \mathsf{Choice}_{214} \xrightarrow{ready} \mathsf{Choice}_{2143}, \ \mathsf{Client}_4\mathsf{(CS)} \colon \mathsf{Checked} \xrightarrow{notYet} \mathsf{Disallowed} \tag{26}$$

$$* \ \mathsf{Server(Evol)} \colon \mathsf{Choice}_{214} \xrightarrow{ready} \mathsf{Choice}_{2143}, \ \mathsf{Client}_4\mathsf{(CS)} \colon \mathsf{With} \xrightarrow{triv} \mathsf{Allowed} \tag{27}$$

$$* \ \mathsf{Server(Evol)} \colon \mathsf{Choice}_{214} \xrightarrow{ready} \mathsf{Choice}_{2143}, \ \mathsf{Client}_4\mathsf{(CS)} \colon \mathsf{Checked} \xrightarrow{request} \mathsf{Allowed} \tag{28}$$

$$\mathsf{Server}\colon \mathsf{Checking}_4 \xrightarrow{pass} \mathsf{Idle} \;*\; \mathsf{Client}_4(\mathsf{CS})\colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Disallowed} \tag{29}$$

$$\mathsf{Server}\colon \mathsf{Checking}_4 \xrightarrow{grant} \mathsf{Helping}_4 \;*\; \mathsf{Client}_4(\mathsf{CS})\colon \mathsf{Allowed} \xrightarrow{triv} \mathsf{Allowed} \tag{30}$$

$$\begin{aligned}
&\mathsf{p}_1 \cdot [\,\mathsf{Server}\colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Helping}_1 \;*\; \mathsf{Client}_1(\mathsf{CS})\colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Allowed}\,] \oplus \\
&\mathsf{p}_2 \cdot [\,\mathsf{Server}\colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Helping}_2 \;*\; \mathsf{Client}_2(\mathsf{CS})\colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Allowed}\,] \oplus \\
&\mathsf{p}_4 \cdot [\,\mathsf{Server}\colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Helping}_4 \;*\; \mathsf{Client}_4(\mathsf{CS})\colon \mathsf{Disallowed} \xrightarrow{triv} \mathsf{Allowed}\,] \oplus \\
&\mathsf{p}_3 \cdot [\,\mathsf{Server}\colon \mathsf{Idle} \xrightarrow{grant} \mathsf{Helping}_3 \;*\; \mathsf{Client}_3(\mathsf{CS})\colon \mathsf{Without} \xrightarrow{triv} \mathsf{Allowed}, \\
&\qquad\qquad\qquad\qquad\quad \mathsf{Server}(\mathsf{Evol})\colon \mathsf{Choice}_{2143} \xrightarrow{ready} \mathsf{ToBe}\,]
\end{aligned} \tag{31}$$

Again, highly similar to the previous two groups of rules, we have three choreography steps addressing the actual phase transfer from Choice_{214} to Choice_{2143} as well as three orchestration steps covering the deviating part of the CS protocol under migration. But here, by taking the last orchestration step, addressing Client_3 for the first time during phase Choice_{2143}, the migration coordination is finished by additionally conduction the phase transfer from Choice_{2143} to ToBe. This then enables McPal to take over the migration coordination, actually by instigating the phase transfer back to Hibernating and the later removal of consistency rules and other model fragments obsolete by then.

5 Adaptation analysis with Prism

We analyze the Paradigm models and their dynamic adaptation with the probabilistic model checker Prism [12]. As it turns out, the Paradigm models involved can conveniently be translated into the Prism modeling language. In particular, each component of the system, Clients, Server and McPal, are interpreted as a separate module. Thus, the detailed behaviour of a component together with its two roles, for the CS collaboration and for the Evol adaptation collaboration, are brought together in one module. This way, the temporary behavioural constraints on the detailed STD imposed by a current phase of the global STDs, can be imposed using a guard on detailed transitions.

A fragment of the Prism specification of Client_1 is shown below. The current state of the detailed STD is stored in local variable s_1. Variables S_1 and E_1 hold, respectively, the current phase of the CS partition and the Evol partition. The Prism fragment specifies the detailed transition of Client_1 as constrained by phase Without of Client(CS), combined with the constraints imposed by phases AsIs and ToBe of the Client(Evol) partition.

$$\begin{aligned}
[\mathtt{enter}_1] \quad & S_1{=}\mathsf{Without} \,\&\, (E_1{=}\mathsf{AsIs} \,|\, E_1{=}\mathsf{ToBe}) \,\&\, s_1{=}\mathsf{Out} \rightarrow s_1'{=}\mathsf{Nosing};\\
[\mathtt{choose}_1] \quad & S_1{=}\mathsf{Without} \,\&\, (E_1{=}\mathsf{AsIs} \,|\, E_1{=}\mathsf{ToBe}) \,\&\, s_1{=}\mathsf{Nosing} \rightarrow s_1'{=}\mathsf{Waiting};\\
[\mathtt{leave}_1] \quad & S_1{=}\mathsf{Without} \,\&\, E_1{=}\mathsf{AsIs} \,\&\, s_1{=}\mathsf{AtDoor} \rightarrow s_1'{=}\mathsf{Out};\\
[\mathtt{leave}_1] \quad & S_1{=}\mathsf{Without} \,\&\, E_1{=}\mathsf{ToBe} \,\&\, s_1{=}\mathsf{AtDoor} \rightarrow \mathsf{q}_1\colon (s_1'{=}\mathsf{Out}) + \mathsf{q}_2\colon s_1'{=}\mathsf{Nosing};
\end{aligned}$$

For the protocol steps within a collaboration, e.g. between the server and its clients, a unique action label identifies a protocol step. The same action label is shared among all components involved. Synchronization on the shared label, hence fulfillment of all relevant guards, leads to execution of the corresponding

consistency rule: a detailed transition of the conductor, phase changes for the participants involved. In this case, the guard indicates that the corresponding trap within a current phase has been entered (at the level of detailed dynamics of the component). In Paradigm, for a phase transfer to be enabled, information about their current states from both the detailed as well as the global STDs, needs to be provided. The information is extracted from the local variables in the Prism module, conjunctively combined as a guard for the global transition. For instance, the unique name of the consistency rule 2 for $i = 1$ is $cr2_{12}$. The consistency rule in Prism is specified by three separate commands, all having the same action label:

In module Client_1: $[cr2_{12}]$ $S_1{=}\mathsf{With}$ & $(s_1{=}\mathsf{AtDoor} | s_1{=}\mathsf{Out} | s_1{=}\mathsf{Nosing}) \rightarrow S_1'{=}\mathsf{Without}$;
In module Client_2: $[cr2_{12}]$ $S_2{=}\mathsf{Without}$ &
$(s_2{=}\mathsf{AtDoor} | s_2{=}\mathsf{Out} | s_2{=}\mathsf{Nosing} | s_2 = \mathsf{Waiting}) \rightarrow S_2'{=}\mathsf{Checked}$;
In module Server: $[cr2_{12}]$ $(ES{=}\mathsf{AsIs} | ES{=}\mathsf{Choice}_2)$ & $r{=}\mathsf{Helping}_1 \rightarrow r'{=}\mathsf{Checking}_2$;

The condition $(s_2{=}\mathsf{AtDoor} | s_2{=}\mathsf{Out} | s_2{=}\mathsf{Nosing} | s_2{=}\mathsf{Waiting})$, for instance, specifies that the current local state of the detailed STD of Client_2 belongs to trap triv of the CS phase Without.

Probabilistic consistency rules are translated into Prism in a slightly different manner. A probabilistic rule is applied in two consecutive stages. During the first stage, the conductor of the rule, in this case the Server, selects the next step to be executed according to the underlying probability distribution. This step is not synchronized with any other participant, in particular the Clients. Once the next step is selected, i.e. a Client to serve is chosen, the Server executes the second part of the rule: it accomplishes the step by conducting, this time synchronized, changing its local state and assigning the phase changes to the participants involved.

We have verified a number of qualitative and quantitative properties of the adapting system.[3]

At any moment during system migration, in any phase, including the source phase AsIs and the target phase ToBe, at most one client will be given service. Let `clients_in_cs` count the number of clients being currently served, i.e. having With or Allowed as global state. Then mutual exclusion can easily be expressed as `clients_in_cs <= 1`.

During any phase, if a client is requesting service, eventually a client will get service. With `one_trying` denoting that a client is in state Waiting, and `one_has_service` denoting that a client is served, this liveness property is expressed as `"one_trying" => P>=1 [ F "one_has_service" ]`.

At any time, in any phase, if a client is requesting a service, then eventually this client will get served. More concretely for Client_1, similar for other clients, this is expressed as: `s1=Waiting => P>=1 [ F (S1=With|S1=Allowed) ]`. Note that implicitly this property also expresses that the functionality to provide

[3] Complete Prism specifications of the collaborative processes in adaptation, as described in Section 4, can be found at `http://www.win.tue.nl/~andova`.

service is never interrupted during adaptation, no matter what dynamics of the server or what trajectory towards target behaviour is taken.

Assuming that the adaptation is triggered, the system will adapt to the target ToBe dynamics: `P>=1 [ F ES=6 & E1=2 & E2=2 & E3=2 & E4=2]`. Here, `ES=6` and `Ei=2`, $i = 1, .., 4$, refer to the completely adapted phases of the server and clients. Moreover, every system component, once adapted to the final stage, does not execute old AsIs behaviour anymore. E.g., for Client_1 we have that `"ToBeOfClient1" => P>=1 [ G !"AsIsOfClient1" ]`.

Various quantitative properties can be checked against the model as well. The results discussed below use a reward structure appropriately defined on the model, assigning a reward of 1 each time the server addresses a client, either for checking or for helping.

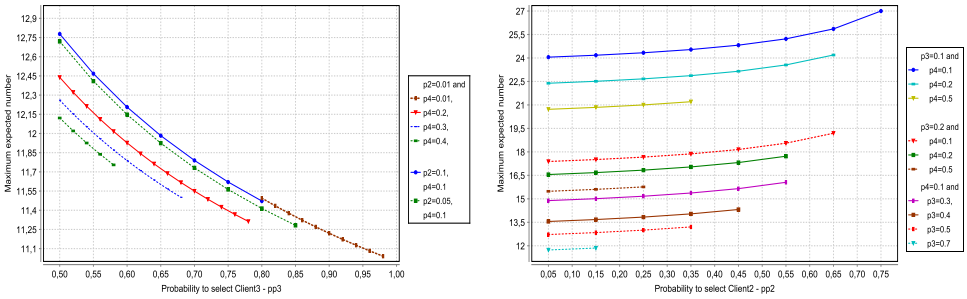

Fig. 11. Maximum expected number of clients addressed by the server during adaptation: fixed p_2 and p_4 (left) and fixed p_3 and p_4 (right)

The expected number of clients addressed by the server during system adaptation is expressed as: `Rmax=? [ F (ES=6) {ES=2} ]`. We compute that for probabilities $\mathsf{p}_1 = \mathsf{p}_2 = \mathsf{p}_3 = \mathsf{p}_4 = 0.25$ this expectation equals 15.3. The experiments show that this actually depends on values of probabilities $\mathsf{p}_1, \ldots, \mathsf{p}_4$. The left graph in Figure 11 shows that the system adapts in less steps as the probability p_3, of selecting Client_3 increases. The right graph in Figure 11 shows that value of p_2, the probability to select Client_2, hardly influences the speed of the system adaptation, but again it is influenced by the value of p_3. From the moment on Client_1 requests service, at any time during the adaptation, the expected number of clients the server addresses before it addresses Client_1 equals 2. We find that the worst case expected number, computed as `Rmax=? [ F (S1=With|S1=Allowed) {s1=Waiting & ES=evol_phase & r=server_state} ]`, gives better insight into the system behaviour during adaptation. The `evol_phase` to be analyzed can be selected, as well as the current local `server_state` of the server, at the moment Client_1 is requesting service. Experiments show that the (worst) expected waiting time for Client_1 decreases as probability p_1 increases, but not for all Evol phases. As expected, for the first two phases, in which Client_1 is not yet selected probabilistically but in round-robin fashion, this measure has

a constant value. Figure 12 shows the results of the experiments for probabilities p_2 and p_3 set to 0.25. As observed, waiting time depends on the current state of the Server at the moment Client_1 requests service. In the graph, r shows the worst case for all particular Evol phases.

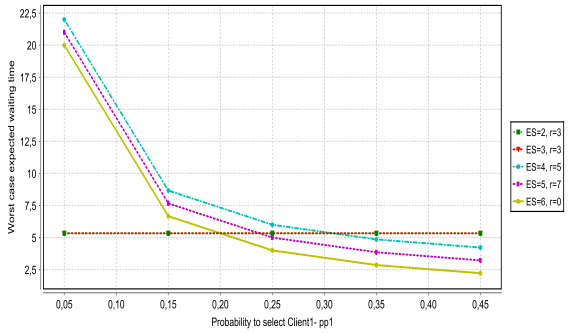

Fig. 12. Worst case waiting time for service for Client_1, for $\mathsf{p}_2 = 0.25$ and $\mathsf{p}_3 = 0.25$

6 Conclusions

We have addressed the issue of formal modeling and analysis, both qualitative and quantitative, of dynamic system adaptation without quiescence. We have shown that Paradigm is well suited to model dynamic adaptation of systems that exhibits probabilistic behaviour. The approach is illustrated for a client-server example. In the source situation, the clients have strictly deterministic dynamics and are served in round-robin fashion. In the target system, clients have probabilistic behaviour, and the server probabilistically selects which client to serve. In the Paradigm model of the adaptation, components smoothly change their behaviour, gradually replacing old deterministic by probabilistic behaviour. The system components migrate from one phase to another, without having their activity disrupted at all.

In addition, a translation to Prism is presented. Each component is represented as a separate module in Prism, synchronizing with other components via shared labels, just as specified by Paradigm's consistency rules. Dynamic constraints typical for Paradigm, whether a phase transfer can take place and whether a local step is allowed, in Prism are specified as command guards, rather straightforwardly. The translation enables the verification of the adaptation model. Transitional properties, both qualitative and quantitative, of the system during the adaptation, are established using the Prism model checker.

As future work we consider the general translation of probabilistic Paradigm into Prism, taking the example presented in this paper as a starting point. Furthermore, we will conduct more case studies of dynamic adaptation of larger systems, involving more intricate probabilities that are expected to mix well with our architectural approach. In particular, we will compare and connect with

the Cactus protocol framework [10], which seems the only other work providing smooth adaptation of distributed systems.

References

1. R. Allen, R. Douence, and D. Garlan. Specifying and analyzing dynamic software architectures. In *Proc. FASE'98*, pages 21–37, 1998.
2. S. Andova, L.P.J. Groenewegen, J. Stafleu, and E.P. de Vink. Formalizing adaptation on-the-fly. In *Proc. FOCLASA'09*, pages 23–44. ENTCS 255, 2009.
3. S. Andova, L.P.J. Groenewegen, J.H.S. Verschuren, and E.P. de Vink. Architecting security with Paradigm. In *Architecting Dependable Systems VI*, pages 255–283. LNCS 5835, 2009.
4. S. Andova, L.P.J. Groenewegen, and E.P. de Vink. Dynamic consistency in process algebra: From Paradigm to ACP. In *Proc. FOCLASA'08*, pages 3–20. ENTCS 229, 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. N. Bencomo, P. Sawyer, G.S. Blair, and P. Grace. Dynamically adaptive systems are product lines too. In *Proc. DSPL 2008*, pages 23–32. Limerick, 2008.
7. M. Bozzano et al. Safety, dependability, and performance analysis of extended AADL models. *The Computer Journal*, 2010. `doi:10.1093/com`.
8. J.S. Bradbury et al. A survey of self-management in dynamic software architecture specifications. In *Proc. WOSS 2004*, pages 28–33. ACM, 2004.
9. C. Cetina, J. Fons, and V. Pelechano. Applying software product lines to build autonomic pervasive systems. In *Proc. SPLC 2008*, pages 117–126. IEEE, 2008.
10. W. Chen, M.A. Hiltunen, and R.D. Schlichting. Constructing adaptive software in distributed systems. In *Proc. ICDCS'01*, pages 635–643. IEEE, 2001.
11. L.P.J. Groenewegen and E.P. de Vink. Evolution on-the-fly with Paradigm. In *Proc. COORDINATION 2006*, pages 97–112. LNCS 4038, 2006.
12. A. Hinton, M.Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H Hermanns and J. Palsberg, editors, *Proc. TACAS 2006*, pages 441–444. LNCS 3920, 2006.
13. J. Kramer and J. Magee. The evolving philosophers problem: dynamic change management. *IEEE Transactions on Software Engineering*, 16:1293–1306, 1990.
14. J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *Proc. FOSE 2007*, pages 259–268. IEEE, 2007.
15. J. Magee and J. Kramer. Dynamic structure in software architectures. *SIGSOFT Software Engineering Notes*, 21:3–14, 1996.
16. B. Morin et al. An aspect-oriented and model-driven approach for managing dynamic variability. In *Proc. MoDELS'08*, pages 782–796. LNCS 5301, 2008.
17. K. Schneider, T. Schuele, and M. Trapp. Verifying the adaptation behavior of embedded systems. In *Proc. SEAMS '06*, pages 16–22. ACM, 2006.
18. J. Zhang and B.H.C. Cheng. Model-based development of dynamically adaptive software. In *Proc. ICSE'06*, pages 371–380. ACM, 2006.
19. J. Zhang, H.J. Goldsby, and B.H.C. Cheng. Modular verification of dynamically adaptive systems. In *Proc. AOSD'09*, pages 161–172. ACM, 2009.