Coordination in Networked Organizations: the Paradigm Approach

Luuk Groenewegen, Niels van Kampenhout

LIACS, Leiden University

Erik de Vink

Dept of Math. and Comp. Sc., TU/e, Eindhoven & LIACS, Leiden University

Abstract

An extension of the coordination specification language Paradigm is presented. In this set-up Paradigm models cater for multiple managers sharing the coordination of a set of common employees. A transition system semantics for the language is provided, that allows for reasoning about such constructions as delegation and self-management in matrix and general network organizations. An elaborated example illustrates the expressiveness of the proposed version of Paradigm.

Key words: Paradigm, coordination, operational semantics, delegation, software architecture

1 Introduction

The coordination specification language Paradigm [SGO87,EG94,GV02] introduces a new approach toward design and analysis of coordination. Paradigm is STD-based in the sense that the various components to be coordinated, each are represented by a separate state transition diagram. In view of the coordination between such STD components, Paradigm offers two central notions, viz. subprocess and trap, in terms of which any concrete coordination is to be expressed.

Intuitively speaking, a subprocess of an STD is a behavioural phase of that STD. A trap of a subprocess is a final stage of that phase such that a trap of a subprocess, once entered, cannot be left within that phase. This allows for the formulation of global behaviour of an STD in terms of a subprocess, a trap thereof, a next subprocess, a trap thereof, etc. Within a global behaviour a trap between two consecutive subprocesses serves as an overlap between the two phases: the trap is indeed trap of the first phase and all of its states are also present as states inside the next phase, be it that there they do not necessarily constitute a trap.

To describe an STD's detailed behaviours consistently with certain global behaviours Paradigm, as presented in [GV02], introduces the employee as a kind of double STD, one of which is the original STD. Actual coordination is then specified in terms of allowed combinations of global behaviours of a set of employees: This gives a sequence of allowed phase combinations and of allowed changes between consecutive phase combinations for these employees. Such sequences correspond with behaviours of another STD, referred to as the manager (of these employees). Employees and managers together constitute a Paradigm model. As it turns out, an STD being a manager of an employee induces a directed is-manager-of relation, called is-mano, between these two STDs: the STD which is the manager, has an is-manager-of relation with the original STD being part of the employee, pointing from the manager STD to the STD within the employee. Stretching the metaphor of manager and employee a little more, the is-manager-of relation can be considered as graph depicting the distribution of responsibility of the organization. Such an organogram is in general a digraph and the organization a network.

Separating combined global behaviour (coordination) from detailed behaviour (local computation) share similarity to ideas behind Manifold [Arb96]. But, Paradigm does allow for bringing coordination and computation together, as the notions of subprocess and trap already give a clear conceptual separation within one component. Compared to approaches using tuple spaces [Gel85], team automata [BEKR01] and statecharts [Har87], Paradigm expresses the coordination not only in terms of immediate communicative steps —informing a different component about what it has to do or about what has been done— but also in terms of explicit longer-term effects: the subprocesses as the phases and the traps as irrevocal stages thereof.

In [GV02] a fully STD-based formulation of Paradigm models has been presented with the restriction that for the consistency between an employee's detailed and global behaviours –the latter in terms of a fixed set of subprocesses and traps– exactly one manager is responsible. This still allows an STD within an employee to participate in multiple is-mano relations with as many managers, each responsible for a different set of global behaviours of the employee. In addition, [GV02] formulates operational semantics for those Paradigm models where each STD is part of at most one employee and where its single manager is a different STD.

However, the kind of coordination problems we want to cover with Paradigm often is more general. We want to reason about situations where STDs are part of more than one employee, thus having separate managers for different sets of global behaviours. We want to allow employees having more than one manager (for the same set of global behaviours) and we want to reason about such situations. We also want to reason about situations where an STD is manager of an employee with its own STD as part, i.e. an STD has an is-mano relation with itself. Therefore the present paper about Paradigm proposes an operational semantics for a larger set of Paradigm models than in [GV02]. To that aim we adapt the original notion of consistency relation into a partial –as opposed to total– form, here referred to as consistency rules, which allows for richer Paradigm models where a set of employees can have more than one manager. In other words, the coordination of one whole set of combined global behaviours is shared among multiple managers. In passing, we present a compact and concise definition of the Paradigm notions, the Paradigm models and their interpretation improving upon earlier expositions.

The larger class of Paradigm models makes Paradigm substantially more suitable for modeling very different kinds of coordination problems, such as communication-based cooperation between software components and/or human components, evolution on-the-fly and mobility: In addition the complete semantics allows one to verify coordination-related properties of all kinds of Paradigm models. Based on the new operational semantics of the extended set of Paradigm models, we present a small but interesting coordination pattern where a manager delegates a part of the coordination to another process.

The structure of our paper is as follows. After the introduction, Section 2 gives a concise definition of the Paradigm concepts, together with an operational semantics. A small example illustrates the concepts and the semantics. Sections 3 contains specific variants of the example, among which the delegation pattern appears. Our experience with SMV for formal, automated verification of coordination properties and a very short impression of partial results established with Paradigm models with respect to evolution on-the-fly is discussed in Section 4.

2 Operational Semantics

In this section we first introduce the basic concepts of Paradigm and illustrate them using a simple production system.

Definition 2.1

(a) A process P is a collection of states $\mathtt{st}(P)$ together with a collection of transition $\mathtt{ts}(P) \subseteq \mathtt{st}(P) \times \mathtt{st}(P)$ between these states.

(b) A subprocess S of a process P is a subset $\mathtt{st}(S) \subseteq \mathtt{st}(P)$ of states together with a subset $\mathtt{ts}(S) \subseteq \mathtt{ts}(P)$ of transitions such that $\mathtt{ts}(S) \subseteq \mathtt{st}(S) \times \mathtt{st}(S)$ and a number of non-empty subsets of $\mathtt{st}(S)$, called traps. If θ is a trap of a subprocess S and $s \to s'$ is a transition in $\mathtt{ts}(S)$, then $s \in \theta$ implies $s' \in \theta$.

(c) A partitioning of a process P is a collection of subprocesses $\{ S_i \mid i \in I \}$ such that

$$\mathtt{st}(P) = \bigcup_{i \in I} \mathtt{st}(S_i) \text{ and } \bigcup_{i \in I} \mathtt{ts}(S_i) \ .$$

Typically, $P(\pi)$ and π denote a partition of the process P.

According to Definition 2.1 a process is identified with its state-transition diagram. For simplicity of presentation we have discarded actions or labels of transitions. The reader will find no difficulty in adding this when desired.

The trap condition states that once a subprocess reaches one of its traps, it can not leave it while the subprocess is active. Control is caught within the trap. Typically, another subprocess will be prescribed with other traps, so that control can move elsewhere. A special trap of a subprocess S is the trivial trap *triv* consisting of all states of S. For technical convenience we will require below that traps of a subprocess are either nested or disjoint.

A particular production unit *PU* can be modeled by the diagram in Figure 2.1.

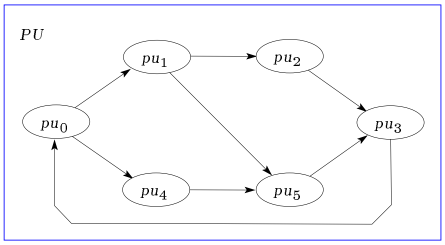


Fig. 2.1. STD Process *PU*

The partition π of *PU* consists of four subprocesses. (See Figure 2.2.) Subprocess *In* has three traps, viz. *triv* (the unnamed outer rectangle), *check* and *done*. Subprocesses *CheckOK*, *CheckKO* and *Out* have besides the trivial trap only one other trap viz, *done*, *check* and *done*, respectively.

Definition 2.2 A Paradigm model $(P_i, \langle \pi_{i,j} \rangle_{j \in J_i}) \mid 1 \leq i \leq n \}$ consists of

- a number of processes $P_1, \ldots, P_n$ with partitions $(\pi_{1,j})_{j \in J_1}, \ldots, (\pi_{n,j})_{j \in J_n}$ for these processes, and
- a collection of consistency rules

  (2.1) $$P \colon s \to s' \ \Leftarrow \ P_i(\pi_{i,j}) \colon S_{i,j} \stackrel{\theta_{i,j}}{\to} S'_{i,j} \qquad i \in I, j \in J_i$$

  where P is one of the processes $P_1, \ldots, P_n$, s, s' are states of P and for each process P_i it holds that $S_{i,j}, S'_{i,j}$ are subprocesses of P_i of the partition $\pi_{i,j}$, $\theta_{i,j}$ is a trap of $S_{i,j}$ containing states of $S'_{i,j}$ only and I is short for the index set $\{ 1, \ldots, n \}$.

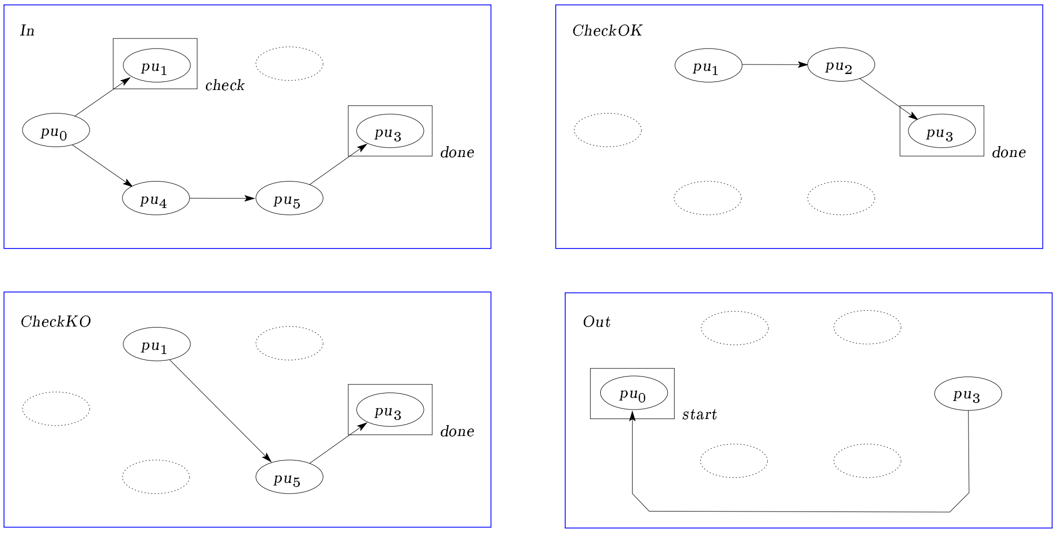


Fig. 2.2. Subprocess *In*, *CheckOK*, *CheckKO*, *Out* of *PU*

A consistency rule of the form (2.1) describes a condition on the local transition $s \to s'$ of the process P. If, for all relevant i and j, the process P_i w.r.t. partition $\pi_{i,j}$ adheres to the subprocess $S_{i,j}$ and resides in some state of P_i that is contained in the trap $\theta_{i,j}$, then P can make the move from s to s'. Additionally, according to this consistency rule, P transfers the process P_i in partition $\pi_{i,j}$ from subprocess $S_{i,j}$ to the subprocess $S'_{i,j}$. Thus, application of a consistency rule does not only affect the local state of the process at the left-hand side, but also influences the local configuration of the other processes as new subprocesses be can be prescribed. Consistency rules express the relationships of local and global behavior: the local transition of P vs. the global transfer of the P_i's from traps $\theta_{i,j}$ to new subprocesses $S'_{i,j}$ for partitions $\pi_{i,j}$.

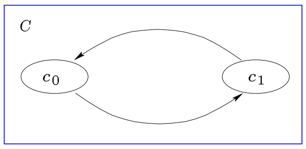
The traps $\theta_{i,j}$ in Equation (2.1) are called connecting traps, viz. connecting the subprocess $S_{i,j}$ and $S'_{i,j}$. As the process P_i does not change state (except for the case were P_i equals P) we require that the current state of P_i is also admitted by $S'_{i,j}$. Whence the condition $\theta_{i,j} \subseteq \mathtt{st}(S_{i,j}) \cap \mathtt{st}(S'_{i,j})$. It turns out, although not strictly necessary, to be good modeling practice to require this trap condition for the situation where P equals P_i as well.

An organizational view at a Paradigm process arises quite naturally. Consistency rules express a one-to-many manager-employee relation. The process P at the left-hand side is the manager, the processes at the right-hand side are the employees. The employees inform the manager that they have reached a certain 'final' stage, i.e. a trap, of the subprocess they are following. There is, in general, no need for the manager to react immediately (as the employees can not leave their traps anyway). The manager synchronizes with its employees and instructs them to continue with the activities of the new subprocesses prescribed.

Two important differences with earlier work on Paradigm should be mentioned here. First, manager-employee relations need not to be hierarchical anymore. In fact, they can change dynamically. For some part of the coordination a process can be the manager, whereas for another part of the coordination the same process can fulfill an employee role. Our formalism allows for maximal flexibility in this respect, hence our mentioning of network organizations generalizing hierarchical ones.

Second, since a process P can have several managers for the same partition, managers are allowed to be partial. They do not need to provide a transfer from a particular subprocess to some other, if another process as manager of the process P may very well cater for this. Dually, two or more managers, can prescribe the same or different transfers in the same situation. Whether this constitutes a bug or a feature is a topic of debate. We have chosen to resolve this via the non-determinism in the operational semantics. Paradigm thus provides a way to distribute responsibilities, not only spatial via separate partitions, but also temporal via applicability of consistency rules.

As an example of a Paradigm model consider the production unit PU introduced above together with a production control C and a quality checker Q. We call this Paradigm model $CPUQ$. The STDs of the processes C and Q are given in Figure 2.3. The global idea is that in state pu_0 the process PU asks the controller process permission to proceed. When permission is granted (subprocess *In*) the production process either goes, at its own choice, in a 'high-quality mode' or or in a 'low-quality mode'. After a first step to pu_1 in high-quality mode the process Q assesses the quality of the product so-far. If OK the production continues in high-quality mode (subprocess *CheckOK*); if not OK the production continues in low-quality mode (subprocess *CheckKO*). When the production unit has reached state pu_5 it waits for permission to return to the state pu_0.



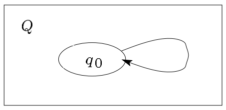


Fig. 2.3. STDs of processes C and Q

We define no partitions for the processes C and Q; process PU has a single partition π with the subprocesses *In*, *CheckOK*, *CheckKO* and *Out* as before. In the model the processes C and Q are managers of the process PU; the process PU is employee of C and Q in its single partition π that is shared by C and Q. It is safe for the processes C and Q to have no partitions and subsequently no subprocesses as these will not occur at the right-hand side of any consistency rule below:

$$C\colon c_0 \to c_1 \Leftarrow PU(\pi)\colon Out \stackrel{start}{\to} In$$
$$C\colon c_1 \to c_0 \Leftarrow PU(\pi)\colon In \stackrel{done}{\to} Out$$
$$C\colon c_1 \to c_0 \Leftarrow PU(\pi)\colon CheckOK \stackrel{done}{\to} Out$$
$$C\colon c_1 \to c_0 \Leftarrow PU(\pi)\colon CheckKO \stackrel{done}{\to} Out$$
$$Q\colon q_0 \to q_0 \Leftarrow PU(\pi)\colon In \stackrel{check}{\to} CheckOK$$
$$Q\colon q_0 \to q_0 \Leftarrow PU(\pi)\colon In \stackrel{check}{\to} CheckKO$$

Note the overlap of the consistency rules of the process Q. The rules only differ in their choice of the new PU-subprocess. This non-determinism is external to PU. In general, such a choice can depend on the local state of the manager (here Q) and the current subprocesses and traps for the other employees (not applicable in the example here).

Definition 2.3 Let $\Pi = \{\, \langle\, P_i, (\pi_{i,j})_{j \in J_i} \,\rangle \mid 1 \leq i \leq n \,\}$ be a Paradigm model.

- A local configuration of the process P in Π is a tuple
$$P[s,\ \langle\, \pi_j(S_j, \theta_j)\,\rangle_{j \in J}]$$
where $(\pi_j)_{j \in J}$ are the partitions of P in Π and each S_j is a subprocess of π_j with smallest trap θ_j containing the state s of P.
- A global configuration of Π is an n-tuple
$$(\ P_i[s_i,\ \pi_{i,j}(S_{i,j}, \theta_{i,j})\,\rangle_{j \in J_i}]\ )_{i \in I}$$
where each component is a local configuration of the particular process of Π and $I = \{1, \ldots, n\}$.
- A global transition of Π consists of two global configurations of Π, notation
$$(\ P_i[s_i,\ \pi_{i,j}(S_{i,j}, \theta_{i,j})\,\rangle_{j \in J_i}]\ )_{i \in I} \to (\ P_i[s'_i,\ \pi_{i,j}(S'_{i,j}, \theta'_{i,j})\,\rangle_{j \in J_i}]\ )_{i \in I}$$
provided that one of the processes P_k of Π has a consistency rule
$$P_k\colon s_k \to s'_k \ \Leftarrow\ P_i(\pi_{i,j})\colon S_{i,j} \stackrel{\hat{\theta}_{i,j}}{\to} S'_{i,j} \qquad i \in I, j \in J_i$$
such that $s_i = s'_i$ for all $i \neq k$, $s_k \to s'_k$ is a transition of $S_{k,j}$, for all $j \in J_k$ and $\hat{\theta}_{i,j}$ is a trap containing the trap $\theta_{i,j}$.

That there is a global transition on the basis of a consistency rule with left-hand side $P_k\colon s_k \to s'_k$ requires that, for each partition π, the current subprocess of π provides the transition $s_k \to s'_k$ in its sub-STD.

A global configuration, also called global state, for the $CPUQ$ Paradigm model introduced above is, for example

$$(\ C[c_1],\ PU[\,pu_0, \pi(In, triv)\,],\ Q[q_0]\ ) \tag{2.2}$$

This configuration reflects the situation where the production unit just got permission from the controller to proceed but has not done so yet. The notation $C[c_1]$ and $Q[q_0]$ reflects that for C and Q no partitions are defined.

The subprocess *In* of PU provides two local transitions for state pu_0. In fact, implicitly consistency rules $P\colon pu_i \to pu_j \;\Leftarrow\;$ for each transition in the STD of process P is in place. As P is playing an employee role only, we do not bother to list these as so-called autonomously consistency rules with empty right-hand side. We have both

$$(\ C[c_1],\ \ PU[\,pu_0, \pi(\mathit{In}, \mathit{triv})\,],\ \ Q[q_0]\ ) \to (\ C[c_1],\ \ PU[\,pu_1, \pi(\mathit{In}, \mathit{check})\,],\ \ Q[q_0]\ )$$

and

$$(\ C[c_1],\ \ PU[\,pu_0, \pi(\mathit{In}, \mathit{triv})\,],\ \ Q[q_0]\ ) \to (\ C[c_1],\ \ PU[\,pu_3, \pi(\mathit{In}, \mathit{triv})\,],\ \ Q[q_0]\ )$$

confirming the internal non-determinism available to the process PU in its state pu_0, either a transition to state pu_1 or a transition to state pu_3. Note the change of trap information of PU in partition π in the first global transition. Also note that no other global transition starting from

$$(\ C[c_1],\ \ PU[\,pu_0, \pi(\mathit{In}, \mathit{triv})\,],\ \ Q[q_0]\ )$$

is possible: C requires PU to have reached to trap *done*; Q requires PU to have reached the trap *check*.

Let us now focus on the global configuration

$$(\ C[c_1],\ \ PU[\,pu_1, \pi(\mathit{In}, \mathit{check})\,],\ \ Q[q_0]\ )\ .$$

The processes C and PU can trigger no transition: C because none of its consistency rules applies, PU because the subprocess *In* provides no local transition from the state pu_0. The process Q comes equipped with two consistency rules that match. This leads to the global transitions

$$(\ C[c_1],\ \ PU[\,pu_1, \pi(\mathit{In}, \mathit{check})\,],\ \ Q[q_0]\ ) \to (\ C[c_1],\ \ PU[\,pu_1, \pi(\mathit{CheckOK}, \mathit{triv})\,],\ \ Q[q_0]\ )$$
$$(\ C[c_1],\ \ PU[\,pu_1, \pi(\mathit{In}, \mathit{check})\,],\ \ Q[q_0]\ ) \to (\ C[c_1],\ \ PU[\,pu_1, \pi(\mathit{CheckKO}, \mathit{triv})\,],\ \ Q[q_0]\ )$$

triggered by the controller process Q. From the perspective of the production unit PU there is external non-determinism, either transfer to the subprocess *CheckOK* or to the subprocess *CheckKO*, caused by the presence of two applicable consistency rules.

The global transition rule of Definition 2.2 transforms the set of global configurations of a Paradigm model into a finite state machine.[1] Endowed with this operational interpretation Paradigm models become amenable to a variety

[1] Only finitely many states are involved in the representation as we choose our processes to be of a finitary nature. It is clear that there is no principle obstacle for the machinery proposed above to deal with infinite state systems.

of formal verification techniques. In Section 4 we discuss some experiments conducted with the model checker SMV.

3 An example

Consider a factory unit for the painting of Delft vases in the colors trendy red for big vases and classic blue for small ones. Vases are presented at three designated spots, S_1, S_2 and S_3, for painting. Transport of vases is dealt with by a part of the factory that is not under consideration. Instead we focus on the painting itself. There are two painting units R and B equipped with an optical sensor that can distinguish big objects from small objects. The painting units are capable of painting one color only. Unit R paints red (for big vases), unit B paints blue (for small vases). See Figure 3.4.

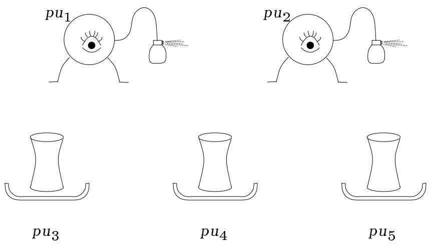


Fig. 3.4. Paintshop 1

We can model this situation in Paradigm by a process R, a process B, and three processes S_1, S_2, and S_3 (cf. Figure 3.5 for the STDs of these processes). Starting from the initial state r_0 the process R can accept a request from either spot. Upon such a request R moves to a state indicating that the request of the particular spot processes has been accepted. From that state R can move back to the initial state after painting of the vase at the spot has been finished. The S processes run as follows: From the starting state s_0 control either moves to the state s_1 or s_3, dependent on the size of the first vase in its queue (s_1 for big vases, s_3 for small ones). After the request for painting has been granted and processed by the respective painting unit, states s_2 and s_4, the process returns to the initial state s_0.

The above description of coordination can be made precise by modeling the above processes as a Paradigm model:

- the processes R and B have partitions ρ and β, respectively with subprocesses *Free* and *Done*,
- the processes S_1, S_2, S_3 have partition σ with subprocesses *Request* and *GetPaint*.

The processes are tied together with the following consistency rules:

$$R\colon r_0 \to r_i \Leftarrow S_i(\sigma)\colon \mathit{Request} \stackrel{\mathit{reqR}}{\to} \mathit{GetPaint}$$

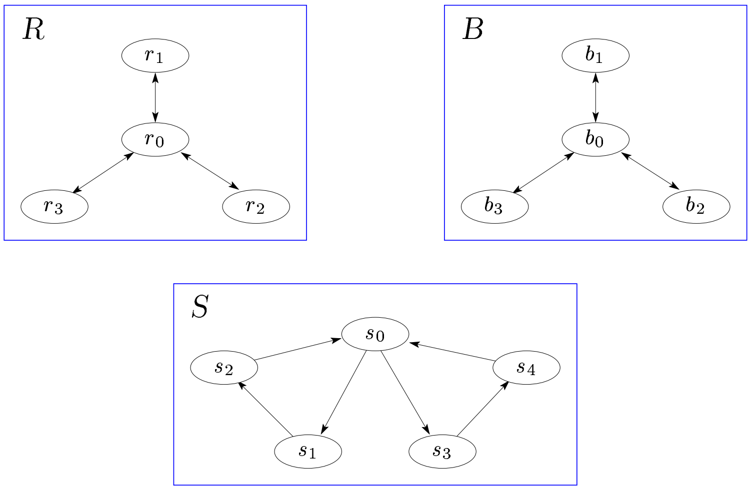


Fig. 3.5. STD R, B and S processes

$$R\colon r_i \to r_0 \Leftarrow R(\rho)\colon \mathit{Done} \overset{\mathit{triv}}{\to} \mathit{Free}$$
$$B\colon b_0 \to b_i \Leftarrow S_i(\sigma)\colon \mathit{Request} \overset{\mathit{reqB}}{\to} \mathit{GetPaint}$$
$$B\colon b_i \to b_0 \Leftarrow B(\beta)\colon \mathit{Done} \overset{\mathit{triv}}{\to} \mathit{Free}$$

for $i = 1, 2, 3$ and

$$S_i\colon s_0 \to s_1 \Leftarrow$$
$$S_i\colon s_0 \to s_3 \Leftarrow$$
$$S_i\colon s_1 \to s_2 \Leftarrow$$
$$S_i\colon s_2 \to s_0 \Leftarrow R(\rho)\colon \mathit{Free} \overset{\mathit{do}(i)}{\to} \mathit{Done},\ \ S_i(\sigma)\colon \mathit{GetPaint} \overset{\mathit{painted}}{\to} \mathit{Request}$$
$$S_i\colon s_3 \to s_4 \Leftarrow$$
$$S_i\colon s_4 \to s_0 \Leftarrow B(\beta)\colon \mathit{Free} \overset{\mathit{do}(i)}{\to} \mathit{Done},\ \ S_i(\sigma)\colon \mathit{GetPaint} \overset{\mathit{painted}}{\to} \mathit{Request}$$

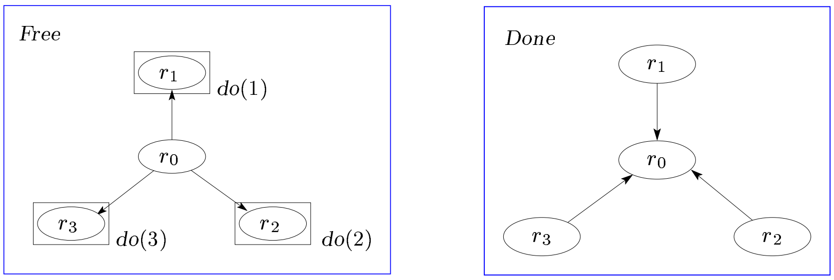


Fig. 3.6. Subprocesses of R (subprocesses for B similar)

The manager-employee or is-mano relation of this of Paradigm model is depicted in the organogram Figure 3.8. Greek letters labeling the arrows indicate the partition to which the manager role applies. Thus, for example both R and B are manager of the spot processes in their single partition σ. Note

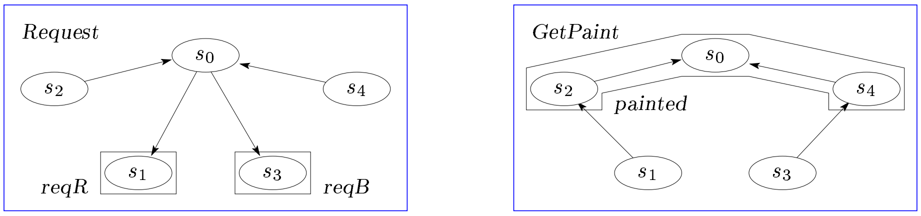


Fig. 3.7. Subprocesses of S

that R and each S_i are both manager and employee with respect to each other. Symmetrically this applies to the process B in relation to the S_i. Self-management occurs, e.g., when R moves from any r_i to r_0 thereby transferring itself from the subprocess *Done* to *Free*. This also applies to the spot processes when returning to the state s_0.

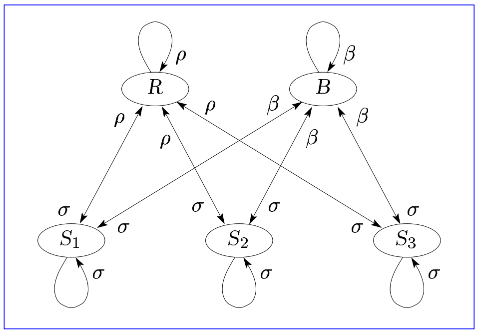


Fig. 3.8. Organogram Paintshop 1

Some consistency rules have an empty right-hand side. Such moves are autonomous as they are independent of other processes and do not result in a transfer of subprocess.

The reader may wonder why the states s_2 and s_4 are included in the subprocess *Request*. The reason is the formulation of the operational rule of Definition 2.3 and the definition of a connecting trap (cf. Definition 2.2). As a spot process transfers itself from *GetPaint* via trap *painted* to subprocess *Request*, the trap *painted* is required to be part of the state space of subprocess *Request*. However, by virtue of the very same consistency rule the spot process will move directly to the state s_0 without residence in neither the included s_2 nor s_4. We could have chosen to relax the requirement on a connecting trap in a situation of self-management as is the the case here. As this will clutter up the basic definitions we did not do so.

The Paradigm model just sketch is unfair in its scheduling as its relies on the scheduling of the non-deterministic choices in the interleaving semantics. For example, from the global state

$$(3.3) \qquad (\ R[r_2, \rho(\mathit{Free}, \mathit{do}(2))],\ B[b_0, \beta(\mathit{Free}, \mathit{triv})],\ S_1[s_0, \sigma(\mathit{Request}, \mathit{triv})],$$
$$S_2[s_2, \sigma(\mathit{GetPaint}, \mathit{painted})],\ S_3[s_1, \sigma(\mathit{Request}, \mathit{reqR})]\ )$$

where at spot S_2 a vase is being painted red. When the spot process is satisfied with the painting it executes the consistency rule $S_2\colon s_2 \to s_0 \Leftarrow R(\rho)\colon \mathit{Free} \overset{do(2)}{\to} \mathit{Done},\ S_2(\sigma)\colon \mathit{GetPaint} \overset{painted}{\to} \mathit{Request}$. Assume that the pending request for red paint of spot process S_3 is not answered by R. Instead, S_2 progresses to its state s_1 for another call upon R on the basis of its autonomous consistency rule and reaches the local configuration $S_2[s_1, \sigma(\mathit{Request}, \mathit{reqR})]$ again; the process R returns to its initial state r_0 in subprocesses *Free* via its self-managing rule. Next, the process R accepts the request of S_2 by executing the consistency rule $R\colon r_0 \to r_2 \Leftarrow S_2(\sigma)\colon \mathit{Request} \overset{reqR}{\to} \mathit{GetPaint}$ reaching the global state 3.3 again. From there the steps described above can be taken once more or, in principle, ad infinitum.

In order to enforce fair scheduling for the painting of red vases in the factory at hand, we decide to adapt the process R (but leave process B as it is). The state r_0 gets split into three copies r_0', r_0'' and r_0'''. In each of these states it is checked in subprocess *Free* whether the spot process associated with the state, i.e. S_1, S_2 and S_3, respectively, is raising a request. If this is the case, the process R moves to the corresponding r_i-state ($i = 1, 2, 3$) and transfers the S_i-process to the *GetPaint* subprocess as before. If the process S_i is not raising a request, the adapted R simply progresses to the next copy of r_0.

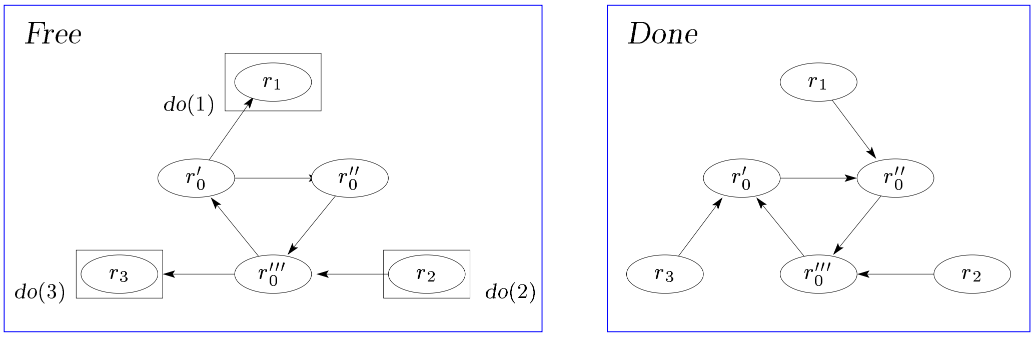


Fig. 3.9. Subprocesses Round-Robin R

We use a negative premise of the form $P(\pi)\colon S \overset{\theta}{\nrightarrow}$ to express that process P in partition π is not executing subprocess S or is executing subprocess S but does not reside in trap θ.

$$R\colon r_0' \to r_1 \Leftarrow S_1(\sigma)\colon \mathit{Request} \overset{reqR}{\to} \mathit{GetPaint}$$

$$R\colon r_0' \to r_0'' \Leftarrow S_1(\sigma)\colon \mathit{Request} \overset{\mathit{reqR}}{\not\to}$$
$$R\colon r_1 \to r_0'' \Leftarrow R(\rho)\colon \mathit{Done} \overset{\mathit{triv}}{\to} \mathit{Free}$$
$$R\colon r_0'' \to r_2 \Leftarrow S_2(\sigma)\colon \mathit{Request} \overset{\mathit{reqR}}{\to} \mathit{GetPaint}$$
$$R\colon r_0'' \to r_0''' \Leftarrow S_2(\sigma)\colon \mathit{Request} \overset{\mathit{reqR}}{\not\to}$$
$$R\colon r_2 \to r_0''' \Leftarrow R(\rho)\colon \mathit{Done} \overset{\mathit{triv}}{\to} \mathit{Free}$$
$$R\colon r_0''' \to r_3 \Leftarrow S_3(\sigma)\colon \mathit{Request} \overset{\mathit{reqR}}{\to} \mathit{GetPaint}$$
$$R\colon r_0''' \to r_0' \Leftarrow S_3(\sigma)\colon \mathit{Request} \overset{\mathit{reqR}}{\not\to}$$
$$R\colon r_3 \to r_0' \Leftarrow R(\rho)\colon \mathit{Done} \overset{\mathit{triv}}{\to} \mathit{Free}$$

The example above where process S_3 could be neglected infinitely often, does not apply anymore. A round-robin scheduling has been enforced upon the model to prevent this. Note that the modification does only affect the process R; the other processes B and S_1, S_2, S_3 remain untouched.

Next we decide to separate the monitoring and painting duties of R and B. We introduce two painter processes P_1 and P_2 both capable of painting red as well as blue. The processes R and B watch the spot processes (R in round-robin fashion, B in an unspecified way). When a request is made by a spot process, R or B instruct an idle painter to paint the particular color at the designated spot. The spot process releases the painter after approval of the painting.

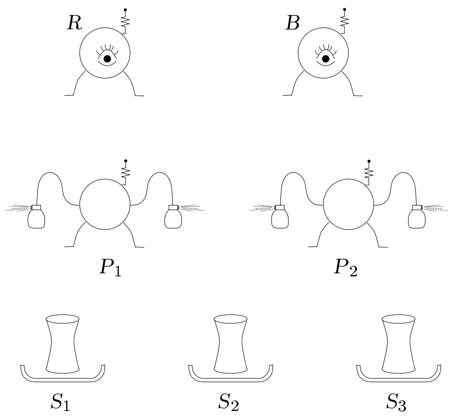


Fig. 3.10. Paintshop 2

The painter processes P_1 and P_2 have the subprocesses as depicted in Figure 3.12. The STD of P_1 and P_2 can be deduced from this subprocesses. Subprocess $\mathit{PaintR}(i)$ for painter P_j corresponds to the situation where the process P_j has been instructed by process R to paint red at spot i. Similar for the blue process B. The STD for R as well as for B need to be adapted. There is no blocking anymore of R or B while a spot process gets painted. The process R, say, has delegated this to the allocated painter process. Therefore R, for the matter of argument, can continue for example with putting the other painter process in charge of painting red at another spot.

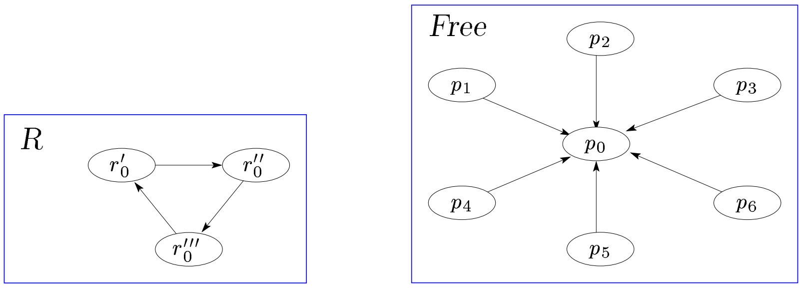


Fig. 3.11. STD modified R and subprocess *Free* of P

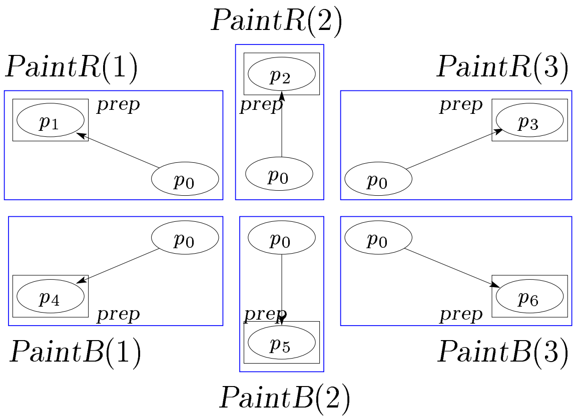


Fig. 3.12. Subprocesses $\mathit{PaintR}(i)$ and $\mathit{PaintB}(i)$ of P

The consistency rules of R, B and S_1, S_2, S_3 will all be modified. This is natural as the painters P_1 and P_2 interact with all these processes. However, the new consistency rules for processes already present follow straightforwardly from the previous ones. New are the rules for the painter process P_j themselves: one rule expressing the autonomy to move to any state p_k $(k = 1, \ldots, 6)$ and a rule to return to the idle state and self-transfer to the *Free*-process in the meanwhile. Note that the applicability of the rules $P_j\colon p_0 \to p_i \;\Leftarrow\;$ is biased: it depends on the subprocess $\mathit{PaintR}(i)$ or $\mathit{PaintB}(i)$ whether such a rule can be fired. This is the side-condition of Definition 2.3c that a local transition is permitted by the current subprocesses.

$$R\colon r_0' \to r_0'' \Leftarrow S_1(\sigma)\colon \mathit{Request} \stackrel{\mathit{reqR}}{\to} \mathit{GetPaint},\;\; P_i(\pi)\colon \mathit{Free} \stackrel{\mathit{idle}}{\to} \mathit{PaintR}(1)$$
$$R\colon r_0' \to r_0'' \Leftarrow S_1(\sigma)\colon \mathit{Request} \stackrel{\mathit{reqR}}{\not\to}$$
$$R\colon r_0'' \to r_0''' \Leftarrow S_2(\sigma)\colon \mathit{Request} \stackrel{\mathit{reqR}}{\to} \mathit{GetPaint},\;\; P_i(\pi)\colon \mathit{Free} \stackrel{\mathit{idle}}{\to} \mathit{PaintR}(2)$$
$$\ldots$$

$$
\begin{aligned}
&B\colon b_0' \to b_0'' \Leftarrow S_1(\sigma)\colon \mathit{Request} \overset{\mathit{reqB}}{\to} \mathit{GetPaint},\ P_i(\pi)\colon \mathit{Free} \overset{\mathit{idle}}{\to} \mathit{PaintB}(1)\\
&\qquad \ldots\\
&S_i\colon s_0 \to s_1 \Leftarrow\\
&S_i\colon s_1 \to s_2 \Leftarrow P_j(\pi)\colon \mathit{PaintR}(i) \overset{\mathit{prep}}{\to} \mathit{PaintR}(i)\\
&S_i\colon s_2 \to s_0 \Leftarrow \mathit{GetPaint} \overset{\mathit{painted}}{\to} \mathit{Request},\ P_j(\pi)\colon \mathit{PaintR}(i) \overset{\mathit{prep}}{\to} \mathit{Done}\\
&\qquad \ldots\\
&P_j\colon p_0 \to p_k \Leftarrow\\
&P_j\colon p_k \to p_0 \Leftarrow P_j(\pi)\colon \mathit{Done} \overset{\mathit{triv}}{\to} \mathit{Free}
\end{aligned}
$$

The organogram underlying the Paradigm model with painter processes is given in Figure 3.13 (partition information omitted). The organization has become layered now with self-management for the S and P processes.

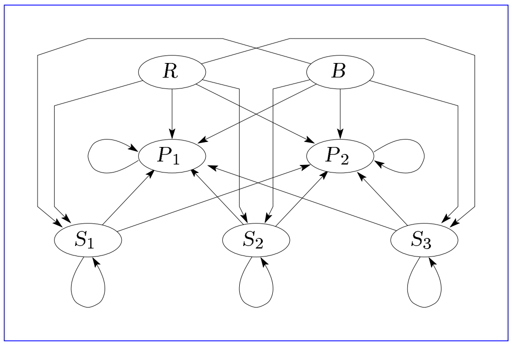


Fig. 3.13. Organogram Paintshop 2

4 Concluding remarks

An extension of the coordination specification languages Paradigm has been discussed. Earlier versions of Paradigm supported hierarchical organizations but with multiple managers in separate partitions. On top of this, the current proposal provides means to express more intricate coordination patterns such as mutual manager-employee relationships, self-management and delegation.

The operational semantics of Paradigm is based on two levels of transitions systems. First there is the local level of STD of the involved Paradigm processes; second there is the global level of subprocesses and traps. The notion of a consistency rule connects the two levels. The semantics of the present paper underlying extended Paradigm also improves upon its conciseness as compared to [GV02].

It should be noted that a modeling of the examples discussed above using multiple partitions is feabsible as well. For example, the processes R and B could have three partitions, one for each spot processes. The Paradigm models presented here are more compact in flavour as having mutiple partitions can be avoided. The drawback is that the models are slightly less transparent as non-interference of managers of the same partitions is not always immediate. However, the view of two-level semantics has paved the way for tool supported experiments exploiting the SMV modelchecker Cadence [McM99]. In the MSc thesis [Kam03] various patterns of delegation has been studied, systematically translated into the SMV language verified using Cadence. It includes a first attempt for reasoning upon delegation. (Independent work on using SMV for the verification of Paradigm models has been reported in [GA02].) Currently our experiments, as based on the present, more liberal coordination regime, suffer from the state-explosion problem. Only relativily small Paradigm models have been inspected yet. However, we hope that standard abstraction and partial evaluation techniques can help us here. Other student work that seeks to relate our approach with the UML has been reported in [Cha02] where in the context of a case study the UML has been endowed with history sensitive information in order to deal with the basic Paradigm notions.

The research reported here has triggered various ideas regarding evolution-on-the-fly. We hope to have convinced the reader that the Paradigm language is flexible in two respects: First, a process can be adapted locally, only requiring its own interface, i.e. its own consistency rules to be modified. Second, new processes can be added and glued into an existing model without much difficulty. New consistency rules define the interaction of a new process and its coordination relation with the processes already present. The general idea for this to be exploited in the context of evolution is as follows: starting from an old Paradigm model and a new desired target Paradigm model, various intermediate Paradigm models can be defined. Change management processes using the trap structure to catch processes that are ready to be updated, are put in place to let the particular process evolve into a new one. In its new incarnation the updated process executes subprocesses of a super-STD of its original one. When an evolution phase has become stable, parts of the earlier behaviour as expressed by parts of the super-STD and related subprocesses can be dispensed with as they will have become unreachable.

A preliminary case-study regarding the classic Dining Philosophers problem that evolves from the basic configuration via deadlock-free stage to a starvation-free situation is coming its way. It should be stressed that both the expressiveness of the Paradigm language as discussed here and the related techniques for semantics based reasoning and verification will make it possible to describe and handle such evolution-on-the-fly schemes.

References

[Arb96] F. Arbab. *Manifold version 2: Language reference manual.* CWI, Amsterdam, 1996.

[BEKR01] M. ter Beek, C. Ellis, J. Kleijn, and G. Rozenberg. Team automata for spatial access control. In *Proc. ECSCW 2001, European Conference on Computer Supported Cooperative Work.* Kluwer, 2001.

[Cha02] M. Chabab. Behaviour and communication in SOCCA and the UML. Master's thesis, LIACS, Leiden University, 2002.

[EG94] G. Engels and L. Groenewegen. Socca: Specifications of coordinated and cooperative activities. In A. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Software Process Modelling and Technology*, pages 71–102. Research Studies Press, 1994.

[GA02] R. Gomez and J. Augosto. Automatic translation of Paradigm models into PLTL-based programs. In *Proc. SEKE02*, pages 497–503. Ischia, ACM Press, 2002.

[Gel85] D. Gelernter. Generative communication in Linda. *ACM Computing Surveys*, 7:80–112, 1985.

[GV02] L. Groenewegen and E. de Vink. Operational semantics for coordination in Paradigm. In F. Arbab and C. Talcott, editors, *Proceedings Coordination 2002*, pages 191–206. LNCS 2315, 2002.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[Kam03] N. van Kampenhout. Systematic specification and verification of coordination: towards patterns for Paradigm models. Master's thesis, LIACS, Leiden University, 2003.

[McM99] K. McMillan. *Getting started with SMV.* Cadence Berkely Labs, 1999. http://www-cad.eecs.berkely.edu/~kenmcmil/tutorial.ps.

[SGO87] M. van Steen, L. Groenewegen, and G. Oosting. Parallel control processes: Modular parallelism and communication. In L. Hertzberger, editor, *Proc. Intelligent Autonomous Systems*, pages 562–579. North-Holland, 1987.