Paradigm as Organization-Oriented Coordination Language

L.P.J. Groenewegen[a], A.W. Stam[a],
P.J. Toussaint[b] & E.P. de Vink[a,c]

[a] *LIACS, Leiden University, The Netherlands*
[b] *LUMC, Leiden, The Netherlands*
[c] *Dept of Math. and Comp. Sc., TU/e, Eindhoven, The Netherlands*

Abstract

Global component behaviours as distinguished in Paradigm, offer the ingredients for specifying inter-component coordination in separation from and consistent with detailed component behaviour. The paper discusses how global behaviours provide great flexibility in arranging computation as well as coordination. In the context of a mediating example we plea for taking such flexibility as an organizational, organic, human-like characteristic; good to have, but usually absent in system specification. In addition, we point out how Paradigm's flexibility fits well in the historical perspective of discrete event simulation, modeling, object-orientation and patterns.

Key words: Paradigm, coordination, organization-orientation, global behaviour, behavioural consistency

1 Introduction

Organizations in the human world always have a goal to reach, a mission to fulfill. To this aim they organize cooperation between their employees, suppliers and customers. Such cooperation is controlled by means of coordination, on the one hand sufficiently precise to guarantee successful cooperation, on the other hand sufficiently flexible to give humans involved enough freedom for initiative, creativity, responsibility and variation. This form of coordination is usually not very strict, often negotiable and implicit. We call this coordination organic, leaving ample room for flexibility. In the world of software, coordination between software components similarly aims at successful cooperation. But here the character of the coordination is completely different: it is rather strict, non-negotiable and explicit. We call this coordination technical.

In this paper we want to contribute to discussing the links between the coordination in two so different worlds, on the basis of our experience with the coordination language Paradigm [6,5]. In this manner we are able to point

out how the rather technical Paradigm notions allow for a new understanding of coordination solutions. Such understanding leads to managerial as well as flexible interpretation and manipulation of coordination solutions giving them some organic flavour. This fits rather well with recent ideas in ArchiMate [9] for instance, to use the same modeling language for the worlds of organizations and of software.

Paradigm, a coordination language for the software world, has been inspired for a large part by organizations too, as from the beginning much attention has been given to coordination in discrete event simulation problems, e.g. from the world of hospitals [12]. Like other coordination approaches, such as Gaia [17], Paradigm keeps the coordination of the components or agents it concerns on a separate, more global level. But rather differently from those other approaches, Paradigm composes this global level from additional global levels of the separate components. Thus, in Paradigm, each component to be coordinated, has its own global levels of behaviour, one for each coordination it is involved in, locally consistent with its underlying detailed behaviour. This *intra*-component consistency is then complemented by *inter*-component consistency on the global coordination level, in Paradigm's case however in terms of the various relevant global levels of the separate components only, thus allowing the coordination to fulfill specific cooperative objectives. As such global coordination level is directly connected to the dynamics of other components, the coordination in Paradigm has similarities to coordination dependency relations between actors as in Tropos [11]. But in Paradigm's case, there always are the two additional globality levels in between: one for each separate component to be coordinated and one for the consistent integration thereof, resulting in consistent underlying detailed dynamics of all components involved.

The intra-component and inter-component consistency together are very useful in solving the dynamic consistency management problems from [8]. This consistency is the topic of [5]. But Paradigm moreover allows for an organic flavour in the above sense, thus constituting a strip of common ground between the two worlds of organization and software coordination. In relation to Paradigm we have found similar strips of common ground between organizations and software, always related to the same difference in character of organic versus technical. This has strengthened our insight into the links we here want to contribute to. Therefore we report on these experiences too.

In the light of the above remarks, the paper has the following structure. Section 2 gives an introduction to Paradigm, pointing out its organizational roots and possibilities in detail, thus providing a completely different interpretation of its notions compared to [6,5]. In Section 3 we line up some of our explorations of what the two worlds have in common. Section 4 then formulates a conclusion and discusses further research.

2 An Organization-Oriented View of Paradigm

Paradigm is a coordination specification language. Its name Paradigm is an acronym of PARallelism, its Analysis, Design and Implementation by a General Method. The idea behind the phrasing General Method is, Paradigm's notions are applicable to rather diverse worlds, ranging from technical ones like operating systems and database management systems via discrete simulation to all kinds of cooperative situations as in businesses, governmental organizations, clubs, families, or even mixtures thereof as in human computer collaboration. For our discussion we repeat the various Paradigm notions from [5] as a listing. For this paper we shall shed a rather different light on them, substantially more organization-coloured than we did elsewhere. In this way we try to bring forward how the Paradigm notions, although perfectly precise and formal and also well-fitting and clarifying in a technical world, nevertheless introduce a certain perceptible flavour of more human-like aspects to coordination, such as managerial aspects as well as organic aspects.

- A process or STD S is a pair $\langle \mathtt{ST}, \mathtt{AL}, \mathtt{TS} \rangle$. Here $\mathtt{ST}$ is called the set of states, or also the state space; $\mathtt{AL}$ is called the set of actions or transition labels, or also the action space; $\mathtt{TS} \subseteq \mathtt{ST} \times \mathtt{AL} \times \mathtt{ST}$ is the set of transitions. We write $x \xrightarrow{a} x'$ in case $(x, a, x') \in \mathtt{TS}$.

- A subprocess of S is a process $\langle \mathtt{st}, \mathtt{al}, \mathtt{ts} \rangle$ such that $\mathtt{st} \subseteq \mathtt{ST}$, $\mathtt{al} \subseteq \mathtt{AL}$ and $\mathtt{ts} = \{ (x, a, x') \in \mathtt{TS} \mid x, x' \in \mathtt{st}, a \in \mathtt{al} \}$.

- A trap t of a subprocess $s = \langle \mathtt{st}, \mathtt{al}, \mathtt{ts} \rangle$ is a nonempty set of states $t \subseteq \mathtt{st}$ such that $x \in t$ and $x \xrightarrow{a} x' \in \mathtt{ts}$ imply that $x' \in t$. If $t = \mathtt{st}$, the trap is called trivial.

- Let $s = \langle \mathtt{st}, \mathtt{al}, \mathtt{ts} \rangle$ and $s' = \langle \mathtt{st}', \mathtt{al}', \mathtt{ts}' \rangle$ be two subprocesses of the same process. A trap t of s is called a connecting trap from s to s' if the states belonging to the trap t are states in s' as well, i.e. $t \subseteq \mathtt{st}'$.

- A partition $\{ (s_i, a_i, t_i) \mid i \in I \}$ of a process $S = \langle \mathtt{ST}, \mathtt{AL}, \mathtt{TS} \rangle$ is a set of subprocesses $s_i = \langle \mathtt{st}_i, \mathtt{al}_i, \mathtt{ts}_i \rangle$ with traps t_i such that $\mathtt{ST} = \bigcup_{i \in I} \mathtt{st}_i$ and $\mathtt{TS} = \bigcup_{i \in I} \mathtt{ts}_i$.

In accordance with the Paradigm notions as defined, whatever coordination problem from the diverse worlds mentioned above, is modeled as follows, descriptively organized so to say.

First, the tasks relevant for what has to be coordinated, are being described. This is done on the basis of purely sequential step sequences, allowing for choices and loops: each such sequence then is a basic task to be performed by one person or machine in (what at first sight seems to be) the same strictly sequential order of the steps as specified. Such basic tasks serve as the units of activity that have to be organized and coordinated in view of some cooperative goal. One basic task description in general allows for many

different realizations or behaviours. Unless explicitly required otherwise, it is assumed each basic task has a unique performer, either person or machine; so it is considered part of the coordination problem if several basic tasks are to be performed by the same performer. Within the Paradigm formalism, such a basic task is specified as a *process*, visualized as a state transition diagram (STD), completely similar to an abstract step-by-step description of a thread of control within object-orientation. Like other sequential descriptions, a basic task may consist of iterations and choices. Infinite descriptions are allowed and also finite descriptions may allow for infinitely many steps to be performed, one after the other. The actually performed, i.e. executed or realized, sequence of steps from a basic task is called a behaviour. So one STD can be considered as a set of (possible) behaviours, each of which can be realized.

Why did we relativize in the above – by '(what at first sight seems to be)' – the equality of the strictly sequential step execution order as realized by the performer and the step order as specified by the corresponding STD? This has to do with the notion of a global (view of the) behaviour of such an underlying, detailed STD. As we shall point out below, the Paradigm notions are such that coordination between processes or STDs is specified in terms of their global behaviours only, suitably chosen in view of the cooperative goal the coordination is aiming at. This allows for some freedom in the execution order of the detailed steps, compared to the strict order of the process specification, as long as on the global level it will make no difference. To be more precise, the steps or transitions of a process correspond to the relevant and detailed actions to be carried out when performing the basic task. The actual sequence of these detailed steps specifies a way, and not necessarily the only one, of how to perform the particular basic task. In this manner, the process specification serves as an underpinning of a kind of contractual requirement, expressing for the cooperation the process performer is involved in, that at any time relevant for the coordination of this cooperation it has to appear as if the actual steps taken are sufficiently in conformity with the corresponding sequence specified. To simplify these matters, one might assume performers to keep strictly to the step sequencing specified in the process description. For software performed by a processor, this is indeed the case. For tasks performed by people this is most often not the case, but as they are held responsible for whatever significant deviation resulting in no longer successful coordination and therefore unsuccessful cooperation, they indeed take usually utmost care to behave sufficiently consistent with what is expected or rather required on the relevant global level.

Second, for the coordination between processes, some global behaviours of these processes are to be defined, depending on the cooperative goal one is after. For the formal description of the global behaviours of a process, Paradigm has the two key notions of *subprocess* and *trap*. A subprocess of a process is a

4

(behavioural) part of that process, a temporary behavioural restriction of the (underlying, original) process: a phase within its full behaviours. This phase is explicitly meant in managerial as well as in an organic sense: during a certain time interval – the duration of the phase – the performer of the basic task remains restricted to certain places or situations or configurations: a suitably chosen subset of the underlying process' states; as the performer should continue behaving, for the duration of the phase the performer moreover restricts its steps to a suitably chosen subset of transitions. Global behaviour of the underlying process then is defined like any behaviour: as a sequence of phases with phase changes in between – completely similar to sequences of states and state changes in between.

For the formal specification of phase changes, Paradigm introduces the *connecting trap* between two subprocesses, which in turn is based of the notion of a trap of a subprocess. A trap of a subprocess is a part of the subprocess' state space, that according to the behaviours of the subprocess, once entered cannot be left. So a performer of a process, behaving in accordance with a subprocess, is trapped in such a trap as soon as a transition to a state of the trap is made. Such a trap then, by its nature, can express an irrevocable stage of the subprocess' behaviours: committing to not leaving the trap for the duration of the subprocess; in addition, such a trap guarantees, the steps within the subprocess preceding the arrival in the trap, are history now, i.e. for the duration of the subprocess. From a managerial point of view, entering a trap is like reaching a milestone on the basis of which some further coordination measures can be taken, such as: investigating whether other performers have reached their milestones, i.e. have entered certain traps; allowing one or more performers ready for it, i.e. after having entered certain traps, to proceed to a next subprocess. So a trap is as a milestone, a global indication: behaviour – of a particular process within the current behaviour restriction of a subprocess – has passed a certain point of no return, on that global level marking the beginning of a final stage of the subprocess. If moreover, for a process residing somewhere in such a trap of its current subprocess, from then on, i.e. on any moment after that trap has been entered, a next subprocess is going to function as the next global behaviour restriction, such a next subprocess should be able to start from any (detailed) state in the trap. On the global level it is then sufficient to know, this can happen independently from the precise state of the trap the (sub)process is in at that particular moment. For this reason Paradigm has the notion of a connecting trap from a subprocess to a new subprocess: the trap is just a normal trap of the old subprocess, such that any of the trap's states also belong to the new subprocess' state space. This is enough to assure the process can start behaving according to the new behavioural restriction from anywhere inside the connecting trap, i.e. independent of (any more detailed knowledge concerning) the precise state it

is in within the trap. Note in addition, the states of the old subprocess' trap generally do not form a trap of the new subprocess, so in general the (former) trap can be left by then.

Particularly the idea of the connecting trap gives Paradigm a usefully organic flavour: on the global level it can remain most unclear where exactly in the detailed behaviour a subprocess changes into a next. It is the strength of the formalism it allows clearly defined phase transitions and phases on the global level – in terms of connecting traps and subprocesses – combined with relative fuzziness on the detailed level: uncertainty to decide on the basis of the detailed behaviour to which phase a state transition actually belongs.
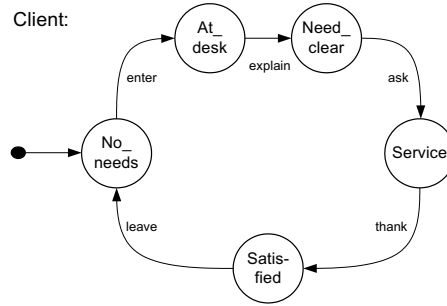
Fig. 2.1. Process Client

We shall now illustrate the above notions and comments with a small example, being a variant of the one in [5]. Figure 2.1 presents the example process of a simple client, either in a shop or in an client-server architecture. A client starts in state No_needs where it can be without connection to any server. Then the client, by entering, goes to state At_desk, where it tries to get in contact with a server via a broker. To this aim the client first explains its wishes by going to state Need_clear. After the broker has understood these wishes, a server is assigned to the client and after in addition the server has the client invited to do so, the client by concretely asking what it wants, gets this service in state Service. Later, by thanking the server, the client shows its satisfaction to the server in state Satisfied whereupon the client, by leaving, returns to state No_needs.

Although a client apparently has rather simple cyclic behaviour, from the more managerial, organizational point of view of a broker or of a server this behaviour still has to many irrelevant details. Therefore we present *partition* Brokering-Serving-Status of a client process, BrSeStatus for short, as visualized in Figure 2.2, consisting of three subprocesses, each specifying a behavioural phase relevant for a broker or a server: WithoutService is the phase where the broker-server organization gives no attention to the client; Orienting is the phase where the client has the attention of the broker; finally, UnderService is the phase where the client has the attention of the server; as in
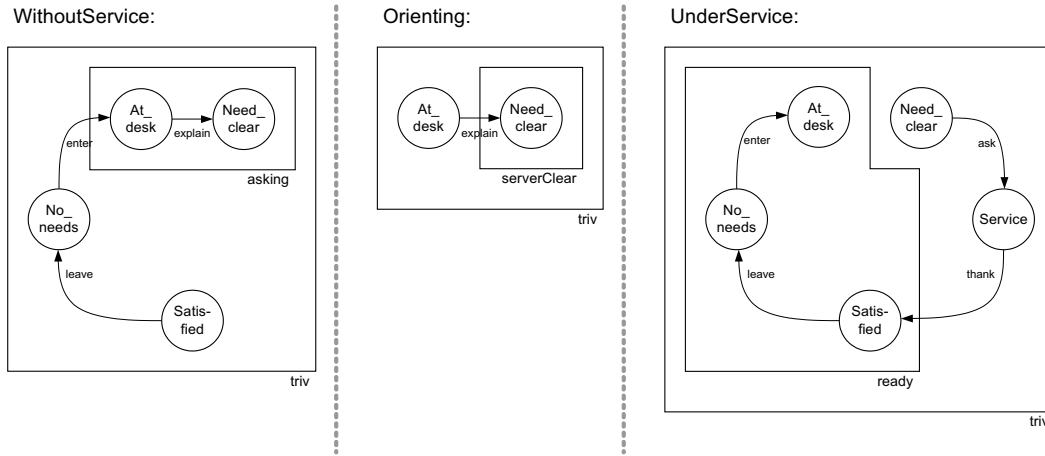
Fig. 2.2. Partition BrSeStatus of a Client process

between the broker's and the (subsequent) server's attention for the client, the client should be in one of these phases too, this is also covered by subprocess Orienting. Note that each subprocess is visualized as an STD, being a part of the original, underlying Client STD; traps are indicated as polygons, containing exactly the states belonging to the trap. Of the traps indicated, we first consider only the nontrivial ones: asking connecting WithoutService to Orienting; questionClear connecting Orienting to UnderService; ready connecting UnderService to WithoutService.
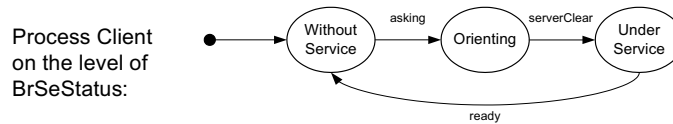

Fig. 2.3. One global behaviour of process Client on the level of partition BrSeStatus

On the basis of the above choice of subprocesses and traps, a global view of the underlying, more detailed client behaviour can be constructed as in Figure 2.3. Again the behaviour is cyclic, here consisting of three steps only instead of the five steps from the detailed description of Figure 2.1. The three nontrivial traps serve in the description as actions. As global behaviours are taking place in the eye of the relevant beholders, the actions on the global level are pure interactions.

One of the relevant beholders of this particular global behaviour is a broker. For this paper it is not the idea to give all details of the Paradigm notions nor of a full-blown Paradigm model. Therefore we shall leave out the servers, but a broker – actually the one broker – is to be presented now, together with some impression of Paradigm's operational semantics. Figure 2.4 visualizes this simple, non-deterministic broker for a fixed number of n clients. In state Check the broker is looking for clients to be brokered. If so, he selects one

according to whatever selection criterion unknown to the modeler. Therefore, in the model the selection is left open as a non-deterministic choice in state Check. In state Mediate(i), with $1 \leq i \leq n$, the one i-th client Client(i) is being brokered.
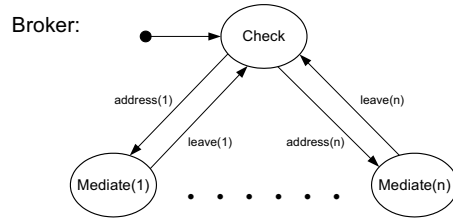

Fig. 2.4. Process Broker

For the behavioural dependencies between the Broker STD and the various client processes Client(i), Paradigm couples one state transition in a so-called manager process, in this case the Broker process, to one or more global subprocess changes for processes (STDs), called employee processes, actually having the relevant global behaviours, in this case the Client(i) processes. Such couplings are called consistency rules, see [5] for a different informal explanation and [4] for a formal definition.

As in a consistency rule exactly one state transition occurs, coupling of state transitions in different STDs is buffered – i.e. asynchronously connected – via the suitably chosen global behaviours: a manager's state transition thus 'prescribes' certain new current subprocesses to some of its employees, exactly corresponding to the induced global transitions occurring in the consistency rule. From then on these employees have to behave according to the newly prescribed subprocesses – whereas before, the same employees had to behave according to the previously prescribed subprocesses. But there is more.

In the global transitions occurring in a consistency rule, always a connecting trap is mentioned as a label for it. This means, the employee it regards, has to have entered that trap (within the currently prescribed subprocess). So a manager has to keep track of the traps currently entered by its employees. Also this is buffered on the level of the relevant global behaviours, be it not via a separate consistency rule, but more implicitly (to be more precise, via the current state of an employee, in combination with its current subprocess on the level of a certain partition, one can straightforwardly decide what the innermost currently entered trap is). As soon as an employee performs a state transition by means of which it actually enters a next, deeper trap of a currently prescribed subprocess, any consistency rule with a global transition labeled with this particular connecting trap, then reaches so to say a higher level of 'being enabled'. Only after all such conditions of a particular consistency rule are fulfilled and the manager arrives in its right state, the rule can 'fire'. Again we see the buffering on the relevant global level: a trap once

entered, cannot be left as long as no new subprocess from the same partition is being prescribed.

Returning to the above example, dependencies – behavioural influencing or also behavioural consistency – between the broker and its clients can be specified by the following consistency rules.

- Broker: Check $\overset{address(i)}{\rightarrow}$ Mediate(i) * Client(i) [BrSeStatus]: WithoutService $\overset{asking}{\rightarrow}$ Orienting

- Broker: Mediate(i) $\overset{leave(i)}{\rightarrow}$ Check * Client(i) [BrSeStatus]: Orienting $\overset{serverClear}{\rightarrow}$ Orienting

So, on the basis of such rules, behavioural consistency between the underlying, detailed processes is dynamically maintained. Note, the above second rule is somewhat incomplete, as the actual delegation towards a particular server is not covered by it; see [5].

In order to point out the organizational, organic flavour of Paradigm, we return to the action labeling a global transition from a subprocess to a next subprocess in terms of a trap. Such an action, an interaction to be more precise, can be formulated on the global level as 'entering that particular trap', thus enabling the global transition. Or it can also be formulated as 'leaving that trap', thus realizing the global transition. Rather more explicitly interaction-like, a reformulation of this interaction can range between 'informing about entering that particular trap' to 'leaving that trap on the basis of new information asked for when the trap was entered'. Here the range of reformulations could mean 'the client sending this information to the broker server organization' or 'the broker server organization receiving this information from the client' or 'the broker server organization sending the client the permission for behaving according to the next subprocess' or 'the client receiving this permission' or even 'the client beginning to use this permission by starting to behave according to the next subprocess'. On the global level this plethora of meanings does not really matter. Whether the interaction is a send or a receive of the question or a send or a receive of the answer or a reacting to as well as in conformity with the answer, is irrelevant. Precisely this makes a behavioural global level description rather managerial, organizational: the differences between sending and receiving, between asking and answering and starting to react to it, are merely technical, irrelevant for the global level where the interaction step only matters as a whole.

In general, the larger a trap, the larger the time interval spanned by the various technical interpretations of the one global step. So, the global level compresses so to say the actual interaction step. On the other hand, some awareness of the (possible) length of this particular time interval still remains: the larger a trap, the larger the time interval that can be possibly spent by executing different detailed steps within the trap, i.e. the more asynchronous is the communication concerning the interaction step labeled by this trap. Commonly one is inclined to reckon the execution time spent within such

9

a trap connecting two subprocesses, to belong completely to the older, first subprocess of the two thus connected, consecutive subprocesses. What gives Paradigm in our opinion undeniably some organic flavour, is the following form of flexibility. It is the freedom to reckon one substantial part of such execution time 'inside a trap' to belong indeed to the first subprocess, but another substantial part of such execution time to belong to the newer, second subprocess of the two.

To give an illustration of such organic flavour – by which we mean, providing room for manoeuvre, flexibility in behaviour – we consider the subprocess change from UnderService to WithoutService via connecting trap ready, as drawn in Figure 2.3. According to the specification from Figure 2.2, two subsequent (state) transitions can occur within trap ready. It is perfectly imaginable, the first transition Satisfied $\overset{leave}{\to}$ No_needs occurs within subprocess UnderService and the second transition No_needs $\overset{enter}{\to}$ At_desk occurs within subprocess WithoutService.
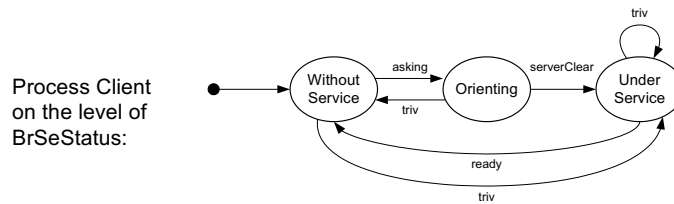

Fig. 2.5. More global behaviour of process Client on the level of partition BrSeStatus

Exactly this phenomenon, of trap behaviour actually occurring (partially) within the second of a pair of consecutively connected subprocesses, is behind the global behaviour specification of Figure 2.5. The specification is in terms of exactly the same subprocesses as before, but now the trivial traps are also taken into account, thus allowing for global behaviour, far more complicated than in Figure 2.3, and what is probably more surprising, substantially more complicated than the underlying detailed behaviour from Figure 2.1.

The kind of global behaviour as in Figure 2.5 being not so much the product of a merely artificial exercise, follows from the concrete illustration hereafter which in this example enables a modeller to adhere quite human-like behaviour to the broker. This feature of the modeling language has a strongly organizational as well as organic flavour: any organization has to cope with such variations of the normal course, a client's global behaviour as in Figure 2.3; any client should have the flexibility, like parts of living organisms, not to let its detailed behaviour be disrupted by such sudden changes in the coordinative control of others, here broker and servers. The illustration is this. All of a sudden, the broker is somewhat in a hurry. So, when during phase Orienting it takes too long before a client has explained its needs, the broker stops its brokering unfinished and let the client return to its previous phase

10

WithoutService, thereby postponing the server selection for it to a later turn of brokering. Even without knowing the behavioural details of the broker, it makes sense to observe, how the trivial trap triv, by being connecting from Orienting to WithoutService, formally enables this return. So the actual state transition At_desk $\overset{explain}{\to}$ Need_clear did not take place while being for the 'first' time within subprocess WithoutService, nor while being within subprocess Orienting either. It might even be postponed until while being within Orienting for another time or until while being within WithoutService for yet another time. So, on the global level the current subprocess restriction for the client can continue to alternate rather frequently, while on the detailed level there is no progression at all.

Analogous freedom exists for adapting the detailed behavioural model of the client STD, as long as the global dynamics on the level of partition BrSeStatus remain unchanged. This is in line with our above remark, relativizing the equality of the step execution order as realized by a performer and the step order as specified by the corresponding STD.

To give an other example illustrating the organizational and organic flavour of Paradigm models, we consider the case arising when the one broker grows ill or when for some other reason the broker has to be away for a while. In such circumstances, the subprocess Orienting is skipped altogether; instead the broker, either beforehand officially and in a well-organized manner or more improvised from the sick-bed, assigns a fixed server to any client, independently from the client's concrete wishes. Then it may happen, a client has not yet articulated its concrete wishes – neither vaguely nor in detail – when its assigned server gives it the serving turn. This then is not to the server's taste, as serving cannot really begin, so it finishes the client's serving turn immediately by letting the client return to phase WithoutService directly, without actually performing any substantial service. Note again the role of the trivial trap triv, connecting from WithoutService to UnderService, thereby enabling the assigned server to give its client the turn even when the client does not yet need it. On the other hand, if some substantial service is given to the client, this can be made visible on the same global level of partition BrSeStatus by means of exactly one transition UnderService $\overset{triv}{\to}$ UnderService per service turn. If moreover the duration of such a service turn is considered relevant, more of such transitions can occur during the same service turn, their number for instance proportional to the duration, thus counting the time spent.

For the technical details of the coordination and delegation as exerted by broker and servers, we refer to [5]. Even when basing our discussion on the technical behavioural specification details of clients only, we have been able to reveal the organizational and organic flavour of the language Paradigm: through its global behaviour descriptions it offers a well-structured fuzziness on top of the strictly specified detailed descriptions, thus allowing for restricted

freedom in the behavioural descriptions. Such freedom, in our opinion, captures some of the essence in organizations of what makes them so human.

3 Charting Common Ground of Software Systems and Organizations

Apart from the organizational and organic flavour of Paradigm, the awareness of which has grown steadily during more than a decade on the basis of intensive modeling experience, there are also many other activities within or related to software engineering revealing different strips of common ground of the two worlds of organizations and software systems. In this section we shall list a few of these and briefly discuss them from the viewpoint of our experience.

Starting point: simulation, discrete event simulation in particular. From the early days of large system programming, simulation programs have been written and executed, for instance SIMSCRIPT and GPSS programs in the early 60s. Such programs always contain a model of the relevant piece of the real world a simulation program user wants to study by letting the model imitate it. At that time it was certainly not common to design such a program by first modeling the program and its execution in detail. Yet, such a model, technical by nature and specifying a technical piece of software, actually would have been a reformulation of the program in some modeling language. Then, being a reformulation, such a model unmistakably would have simultaneously been a model of the relevant real world part (to be) simulated. *Hence, the better a software modeling formalism succeeds in being well suited for specifying all kinds of programs, both in their structure and in their dynamics, the closer the formalism comes to specifying any real world part as good as a particular simulation program for it.*

As a matter of fact, both [15] and [12] used Paradigm modeling successfully for simulation purposes: the former for simulating a large multi-processor variant of an existing UNIX version, being a merely technical system, the latter for simulating a large hospital process around X-ray imaging and X-ray image storing and communicating. No essential differences between technical and human worlds were being encountered concerning the coordination and behavioural consistency modeled, exactly as it was expected.

From simulation to object-orientation. The general, Algol60-like programming language Simula, with special features for facilitating discrete event simulation, has been introduced in the late 60s. Retrospectively one can say, with [10], Simula is the first object-oriented programming language, already long before the term OO had been coined. In line with the above, one can actually argue on the basis of Simula, it is the object-orientedness in particular that is geared towards not just modeling software but also modeling organizations, business processes, human collaborations and mixtures thereof like human-

computer interaction. See also [7] for different but related reasons supporting the same conclusion.

A major problem with object-orientation and its current de facto state-of-the-art standard UML 2.0 is however, it is lacking in behavioural consistency, see e.g. [8]. Nevertheless, this is the main stream state-of-the-art in modeling. *Despite certain shortcomings, object-oriented modeling is as suited for organizations as it is for software, in the sense of accurately reflecting structure in terms of smaller units, the local behaviour of such units, their general flow of activities and their interaction scenario's and compositions thereof.* The readability of such specifications however is another big problem, particularly so for people less educated in ICT.

From object-orientation to patterns. On the basis of the re-occurrence, again and again, of strongly similar constellations of cooperating objects in many different software systems, the idea of a pattern has been adopted from the world of edifice architecture. Since [3] and [1] these ideas have been extended rather pliably towards organization and business process modeling, see [2]. In particular, the typically architectural property of a pattern: determining a global building block without being explicit about its inner technical details, gives a pattern useful flexibility, placing it at a modeler's disposal for quite different contexts. This flexibility and pliable ease of use, is what, compared to procedures, functions, services, gives a pattern its organic flavour. *Thus, patterns by their nature reinforce and connect the organic similarities between the two worlds of software and organizations.*

From patterns to delegation. One aspect as yet remaining under-exposed in the usual pattern descriptions, is their interactions. Often not more than sequence diagrams show some main scenario's. Thorough analysis and proofs of collaborative properties among the pattern's participants are normally not given. This is not really surprising in the light of the above mentioned shortcoming of UML concerning behavioural consistency. The paper [5] together with the Master's Thesis [14] show how Paradigm modeling provides a way out. As an extra feature of these Paradigm solutions, it is shown how easily one can vary these Paradigm models, giving room for improvisation of the solution details depending on the circumstances. *Thus, Paradigm models are in line with normal pattern flexibility.*

From simulation, OO and business process modeling to information system integration. Partly out of familiarity with [12] concerning Paradigm's expressiveness in modeling hospital processes and the importance of having hospital information systems well-integrated into such processes, in [13] the topic of integrating hospital information systems is studied from the broadening perspective of taking the (relevant) hospital processes into account from the beginning. Hence, the integration was not just redoing of (more) code integration, even not just redoing of (more) design integration, but really re-

doing requirements integration. In addition, it was found essential to model the relevant hospital processes, 'connected' to the integrated (software) information system to-be, as explicitly as the software design. Analogous to simulation models, only in this manner possible impacts of the software to-be on the hospital's business processes could be studied in sufficient detail. Again it appeared, *software and surrounding business can be modeled in the same object-oriented modeling language with enough expressiveness for coordination details.* Moreover, in view of the need of redoing requirements integration, *it turned out wise to redo even the requirements engineering, thus engineering the specification of the integrated software system to-be from the integrated requirements.* Together this has led to the following insight. *From the beginning of the requirements engineering process, the two worlds of software and of surrounding organization are to be modeled in one and the same language, in order to be able to analyze their mutual impact, not only their structural impact but also their dynamical impact. During the later software engineering phases such double model should gradually grow towards one well-integrated model.* Both in our teaching and in our work we have found this idea of *integration-orientation,* as we called it, very valuable.

From business process modeling and OO to ArchiMate. To achieve better integration between usually large and complicated software systems and organizations supported by them, architectural descriptions of both are used. On the global level of such a description one has main parts and main streams, the latter being global work flows inside a part like activity sequences or global flows between parts like high level protocol(-role)s. The idea then is, to see on that particular level where, what and why there is some misfit, or to estimate on that particular level impact (regarding certain properties and aspects) of a concrete change somewhere in either architectural description. In this architectural world it was felt as unnecessarily complicating, architectural descriptions of software applications, of infrastructural hardware and of the business world do not match, being written normally in too different languages. To improve the situation in these respects substantially, the ArchiMate project partners devised a meta architectural language, see [9] comprising three levels – business, application and technical – each with its own specialization of the meta language. In this way sufficient cohesion between the domains of the different levels is achieved, while for each level the particular model formulations remain geared towards the peculiarities and preferences reigning on that level. The idea of the same meta language with differently specialized languages for the various levels, is basically the same as the above mentioned integration-orientation: *via the same language only, the different worlds of software and organization can be connected to be studied in sufficient cohesion.* In [16] it has been additionally shown how ArchiMate models can be translated into UML models. It so happens, these UML models have an uncommon degree of

'declarativity', keeping the global character of the original architectural models. This globality is quite unusual, as one is not inclined to allow it when starting to model such architectures from scratch in UML. On the other hand it is clarifying too, as it shows what indeed matters less for a system architect or not at all for a manager.

Although not incorporated within ArchiMate as being non-standard, Paradigm seems to be well-suited here too, particularly for analyzing behavioural consistency and for clarifying behavioural impact details in case of a change.

4   Conclusions and Future Research

We have demonstrated how Paradigm's notions of subprocess and trap allow for organic flexibility in behavioural descriptions on a global level, independent from other behavioural descriptions of the same component. In Paradigm, specifying coordination within a particular pattern can be conveniently mixed with coordination outside that pattern. We see the above flexibility of Paradigm as supplementary to the architectural expressiveness patterns already offer.

Very recent results, not yet published, have been obtained, showing how a given Paradigm model without any halting, can be JIT – Just-In-Time – extended with new semantics, whereupon the model continues with migrating towards a new model. This is evolution on-the-fly by means of self-adaptation; migration then is coordinated as smoothly as desired, in accordance with the semantic details in the JIT-extension. The whole procedure can be iterated, thus enabling yet another migration to a next model. In our opinion, the feature of self-adaptation and the possibility of arranging a migration as smoothly and gracefully as desired, underlines the organic and organizational potential of Paradigm even more.

Future research on more examples of such JIT-modeled migration and evolution is to be carried out, revealing more about character and architecturability of these recent ideas.

References

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture. A System of Patterns.* Wiley, 1996.

[2] J. Coplien and N. Harrison. *Organizational Patterns of Agile Software Development.* Prentice-Hall, 2004.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[4] L. Groenewegen, N. van Kampenhout, and E. de Vink. Coordination in Networked Organizations: the Paradigm Approach. Technical Report CSR 03/13, Technische Universiteit Eindhoven, 2003.

[5] L. Groenewegen, N. van Kampenhout, and E. de Vink. Delegation Modeling with Paradigm. In *Proc. Coordination '05, Namur*, 2005 (To appear).

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

[7] I. Jacobson, M. Ericsson, and A. Jacobson. *The Object Advantage: Business Process Reengineering with Object Technology.* Addison-Wesley, 1994.

[8] J. Kuester. *Consistency Management of Object-Oriented Behavioral Models.* PhD thesis, University of Paderborn, 2004.

[9] M. Lankhorst, editor. *Enterprise Architecture at Work: Modelling, Communication and Analysis.* Springer, 2005 (To appear).

[10] B. Meyer. *Object-Oriented Software Construction.* Prentice-Hall, 1988.

[11] A. Perini, A. Susi, and F Giunchiglia. Coordination specification in multi-agent systems: From requirements to architecture with the Tropos methodology. In *Proc. SEKE 2002, Ischia, Italy*, pages 51–54. ACM, 2002.

[12] W. Stut. *Constructing Large Conceptual Models with Movie.* PhD thesis, Leiden University, 1992.

[13] P.J. Toussaint. *Integration of Information Systems: a Study in Requirements Engineering.* PhD thesis, Leiden University, 1998.

[14] N. van Kampenhout. Systematic Specification and Verification of Coordination: towards Patterns for Paradigm Models. Master's thesis, LIACS, Leiden University, 2003.

[15] M. van Steen. *Modeling Dynamic Systems by Parallel Decision Processes.* PhD thesis, Leiden University, 1988.

[16] M. Wiering, M. Bonsangue, R. van Buuren, L. Groenewegen, H. Jonkers, and M. Lankhorst. Investigating the Mapping of an Enterprise Description Language into UML 2.0. In F.de Boer and M.Bonsangue, editors, *Proc. UML 2003 Workshop on Compositional Verification of UML Models.*, volume 101 of *ENTCS*, pages 155–179, 2004.

[17] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering Methodology*, 12:317–370, 2003.