# Towards reduction of Paradigm coordination models

Suzana Andova*
Department of Mathematics and Computer Science
TU/e, Eindhoven, the Netherlands

Luuk Groenewegen
FaST Group, Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands

Erik de Vink
Department of Mathematics and Computer Science
TU/e, Eindhoven, the Netherlands

**Abstract** The coordination modelling language Paradigm addresses collaboration between components in terms of dynamic constraints. Within a Paradigm model, component dynamics are consistently specified at a detailed and a global level of abstraction. To enable automated verification of Paradigm models, a translation of Paradigm into process algebra has been defined in previous work. In this paper we investigate, guided by a client-server example, reduction of Paradigm models based on a notion of global inertness. Representation of Paradigm models as process algebraic specifications helps to establish a property-preserving equivalence relation between the original and the reduced Paradigm model. Experiments indicate that in this way larger Paradigm models can be analyzed.

**Keywords** coordination, process algebra, Paradigm, vertical dynamic consistency, levels of abstraction, branching bisimulation, globally inert, model reduction

## 1 Introduction

Within the current software architecture practice, architectures are mostly used for describing static aspects of software systems. Techniques that allow system architects to describe coordination among components within an architecture and to reason about the dynamics of the system in its entirety, are not commonly used. The coordination description language Paradigm helps the designer to merge different dynamic aspects of a system. At the same time the language allows for the description of both detailed and global behaviour of an individual component i.e. its own specific behaviour and separately its interaction with other components, and the language is particularly helpful in enforcing consistency in the behaviour of large sets of interrelated components.

The coordination modeling language Paradigm [9, 10] specifies roles and interactions within collaborations between components. Interactions are in terms of temporary constraints on the dynamics of components. To underpin Paradigm models with formal verification and automated analysis, the Paradigm language has been linked with the `mCRL2` toolset [11] via its translation to the process algebra ACP [6, 3] and with the probabilistic modelchecker Prism [15, 4] via a direct encoding scheme. Process algebras (PA for short), such as CCS, CSP, LOTOS and ACP, provide a powerful framework for formal modeling and reasoning about concurrent systems, which turns out to be very suitable for our needs in the setting of coordination. The key concepts of compositionality and synchronization in process algebra are mostly exploited in our translation. As detailed and global aspects of component behaviour are specified by separate PA specifications, the vertical constraints are encoded through synchronizations expressing consistency of detailed and global component behaviour. Horizontal constraints at the protocol level are naturally captured by parallel composition, synchronization and encapsulation.

*Corresponding author, email `s.andova@tue.nl`.

To appear in EPTCS.

While the translation to ACP and mCRL2 allows for formal verification of Paradigm models [3, 2, 4], the omnipresent problem of state space explosion when analyzing large models occurs here as well. In the present paper, we address the question of reducing Paradigm models of coordination. The reduction method applies to a component's behaviour, reducing the representation of the vertical constraints of that component by abstracting away any information on the component behaviour irrelevant for these constraints. To this end, the benefit of the translation of Paradigm language into ACP is twofold. On the one hand, we borrow the abstraction concept from PA and apply it directly in Paradigm on detailed behaviour. On the other hand, the translation provides us with a formal proof methodology to reason and guarantee that the reduced Paradigm model has the same properties as the original model. As a matter of fact, it has gradually become evident that separating detailed from global behaviour as supported by the Paradigm language, allows us to reason about reduction by abstraction in a rather natural way. We shall clarify this point after the Paradigm overview, at the end of Section 2.

Our work on dynamic consistency in a horizontal and vertical dimension has been influenced by the work of Küster [7, 14]. Related work includes the Wright language [1] based on CSP provides FDR support to check both types of consistency properties. Other bridges from software architecture to automated verification include the pipeline from UML via Rebeca and Promela to the SPIN model-checker and from UML via Object-Z and CSP to the FDR model-checker [19, 16]. Process algebra driven prototyping as coordination from CCS is proposed in [18]. The skeletons generated from CCS-specifications overlap with Paradigm collaborations. In the TITAN framework [17], CCS is playing a unifying role in a heterogeneous environment for aspect-oriented software engineering. Recently the coordination language Reo has been equipped with a process algebraic interpretation [5, 12]. The encoding of Reo into mCRL2 and subsequent analysis has been integrated in the ECT toolset for Reo [13].

We present our idea by means of an example. The system we consider consists of $n$ clients who try to get service from one server exclusively, a critical section problem, where the server is supposed to choose the next client in a non-deterministic manner. While the translation of the Paradigm model into PA for the example is done manually, the toolset mCRL2 is exploited to generate the complete state spaces, on which further analysis can be done. Initial results show a substantial reduction in the size of the state space. In Section 2 Paradigm is summarized on the basis of the above example. Section 3 briefly introduces our process algebra translation for the example model. In Section 4 we present our reduction techniques. Section 5 concludes the paper.

## 2 Paradigm and a critical section model

This section briefly describes the central notions of Paradigm: STD, phase, (connecting) trap, role and consistency rule.

- An *STD* $Z$ (*state-transition diagram*) is a triple $Z = \langle \mathtt{ST}, \mathtt{AC}, \mathtt{TR} \rangle$ with $\mathtt{ST}$ the set of states, $\mathtt{AC}$ the set of actions and $\mathtt{TR} \subseteq \mathtt{ST} \times \mathtt{AC} \times \mathtt{ST}$ the set of transitions of $Z$, notation $x \xrightarrow{a} x'$.
- A *phase* $S$ of an STD $Z = \langle \mathtt{ST}, \mathtt{AC}, \mathtt{TR} \rangle$ is an STD $S = \langle \mathtt{st}, \mathtt{ac}, \mathtt{tr} \rangle$ such that $\mathtt{st} \subseteq \mathtt{ST}$, $\mathtt{ac} \subseteq \mathtt{AC}$ and $\mathtt{tr} \subseteq \{ (x,a,x') \in \mathtt{TR} \mid x,x' \in \mathtt{st}, a \in \mathtt{ac} \}$.
- A *trap* $t$ of phase $S = \langle \mathtt{st}, \mathtt{ac}, \mathtt{tr} \rangle$ of STD $Z$ is a non-empty set of states $t \subseteq \mathtt{st}$ such that $x \in t$ and $x \xrightarrow{a} x' \in \mathtt{tr}$ imply $x' \in t$. A trap $t$ of phase $S$ of STD $Z$ *connects* phase $S$ to a phase $S' = \langle \mathtt{st}', \mathtt{ac}', \mathtt{tr}' \rangle$ of $Z$ if $t \subseteq \mathtt{st}'$. Such trap-based connectivity between two phases of $Z$ is called a *phase transfer* and is denoted as $S \xrightarrow{t} S'$.

- A *partition* $\pi = \{ (S_i, T_i) \mid i \in I \}$ of an STD $Z = \langle \mathtt{ST}, \mathtt{AC}, \mathtt{TR} \rangle$, $I$ a non-empty index set, is a set of pairs $(S_i, T_i)$ consisting of a phase $S_i = \langle \mathtt{st}_i, \mathtt{ac}_i, \mathtt{tr}_i \rangle$ of $Z$ and of a set $T_i$ of traps of $S_i$.
- A *role* at the level of a partition $\pi = \{ (S_i, T_i) \mid i \in I \}$ of an STD $Z = \langle \mathtt{ST}, \mathtt{AC}, \mathtt{TR} \rangle$ is an STD $Z(\pi) = \langle \widehat{\mathtt{ST}}, \widehat{\mathtt{AC}}, \widehat{\mathtt{TR}} \rangle$ with $\widehat{\mathtt{ST}} \subseteq \{ S_i \mid i \in I \}$, $\widehat{\mathtt{AC}} \subseteq \bigcup_{i \in I} T_i$ and $\widehat{\mathtt{TR}} \subseteq \{ S_i \xrightarrow{t} S_j \mid i, j \in I, t \in \widehat{\mathtt{AC}} \}$ a set of phase transfers. $Z$ is called the *detailed* STD underlying *global* STD $Z(\pi)$, being role $Z(\pi)$.
- A *consistency rule* or *protocol step* for an ensemble of STDs $Z, Z_1, \ldots, Z_k$ and roles $Z_1(\pi_1), \ldots, Z_k(\pi_k)$ is a nonempty set of phase transfers preceded by one extra transition.
- Let $Z \colon x \xrightarrow{a} x' * Z_1(\pi_1) \colon S'_1 \xrightarrow{t} S''_1, \ldots, Z_k(\pi_k) \colon S'_k \xrightarrow{t} S''_k$ be a consistency rule for a given ensemble; $Z_i, \ldots, Z_k$ are *participants* of it, $Z$ is *conductor*.
- A Paradigm *model* is an ensemble of STDs, roles thereof and consistency rules.

The above notions constitute Paradigm models. The semantics thereof are roughly as follows: a consistency rule has synchronization of its phase transfers and its conductor transition, only if all connecting traps mentioned have been entered. Detailed transitions are allowed in the current state of an STD, only if the current phase (state) of each role of the STD contains the transition. In this way, phases are constraints on underlying STD dynamics imposed by protocols (sets of protocol steps). In a mirrored way, traps impose constraints on the behaviour at the protocol level, as traps are involved in the firing of consistency rules.

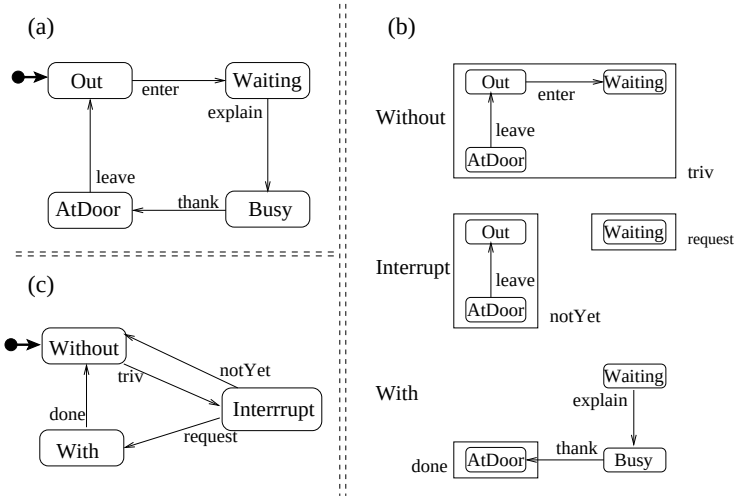


Figure 1: (a) detailed STD of Client, (b) partition of three phases, (c) global STD Client(CS).

An STD is a step-wise description of the dynamics belonging to a component. It is visualized as a directed graph: its nodes are states, its action-labeled edges are transitions. Initial states are graphically indicated by a black dot-and-arrow. Figure 1a gives the so-called detailed STD of a Client in and around a shop: starting in state Out the client cycles through states Waiting, Busy, AtDoor and Out again, subsequently. The entire system we consider, contains $n$ such clients, dynamically the same, plus one different component, the server. For the complete system the overall requirement is that only one client at a time, out of all $n$ clients, is allowed to be in its state Busy. So, being in state Busy is a Critical Section problem (abbreviated CS). To solve it, ongoing $\mathsf{Client}_i$ dynamics is constrained by the phase prescribed currently. Figure 1b visualizes phases Without, Interrupt and With. Phase Without excludes being in state Busy by prohibiting to take the actions explain and thank. Contrarily, phase With allows

both, going to and leaving state Busy. Finally, the intermediate phase Interrupt is an interrupted form of Without, as action enter cannot be taken, but being in state Waiting is allowed, though.

In view of a transfer from the current phase into a next phase to occur, enough progress within the current phase must have been made: a connecting trap has to be entered first. Figure 1b pictures relevant connecting traps for the above three phases, drawn as rectangles around the states the trap consists of. In particular, we need trap triv to be connecting from Without to Interrupt, trap notYet to be connecting from Interrupt back to Without, trap request to be connecting from Interrupt forward to With and finally, trap done to be connecting from With back to Without. In this manner, Figure 1b gives all ingredients needed for the dynamics of a $\mathsf{Client}_i$ STD at the level of partition CS: see role $\mathsf{Client}_i(\mathsf{CS})$ in Figure 1c and repeated in Figure 2a.

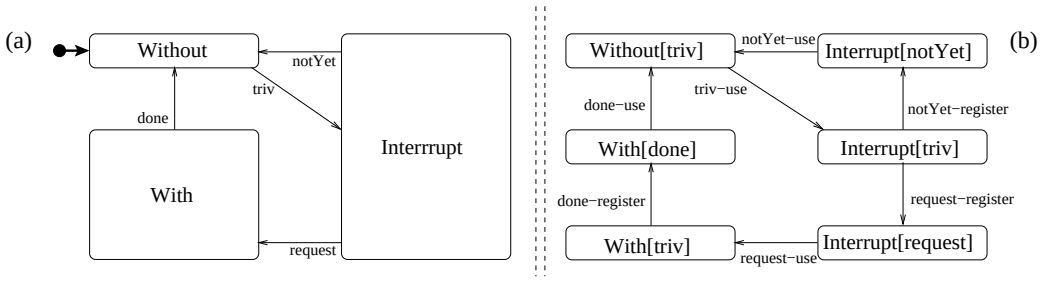


Figure 2: (a) global process $\mathsf{Client}(\mathsf{CS})$ and (b) its refinement in view of translation.

Figure 2b presents a slightly refined diagram of the proper role STD in part (a). State names here, additionally keep track of the trap most recently entered within a phase, as if it could be taken as a smaller phase committed to within the larger one imposed. Action names still refer to a trap that is entered, but they additionally discriminate between, first, *registering* the trap has been entered and, second, thereafter *using* this for a phase transfer. This more refined view represents the starting point for the ACP encoding of the global process, as discussed in the next section.

So far, we have discussed 'sequential composition' of constraints: imposed phases alternated with traps committed to. Semantically, any current phase constrains the enabled transitions to those belonging to the phase. So, at any moment a current detailed state belongs to the current phase too. From this it follows, that the dynamics of the detailed STD and of the global STD are consistent, the current global phase reflects the current local state. Paradigm's consistency rules are to the essence of 'parallel composition': they express coupling of role steps of arbitrarily many *participants* and a detailed step of one *conductor*. Any consistency rule specifies the simultaneous execution of the steps mentioned in the rule, a transition of the conductor and phase transfers for the participants.

To continue the example of $n$ clients getting service, one at a time, we present a non-deterministic coordination solution for the $n$ clients via a server. The non-deterministic server checks the clients in arbitrary order. If a client, when checked, wants help, it gets help by being permitted to enter the critical section. If not, permission to enter is refused to it. Only after a client's leaving the critical section, the server stops helping it by returning to the idle position, from which it arbitrarily selects a next client for checking. In the example, the server provides a unique conductor step for each consistency rule. The STD Server of the server is drawn in Figure 3. As conductor, detailed steps of Server need to be coupled to phase transfers of each $\mathsf{Client}_i$, $1 \leq i \leq n$.

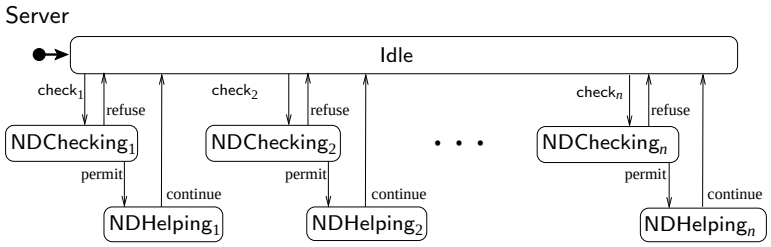


Figure 3: STD non-deterministic server Server.

$$\mathsf{Server} : \mathsf{Idle} \xrightarrow{\mathit{check}_i} \mathsf{NDChecking}_i * \mathsf{Client}_i(\mathsf{CS}) : \mathsf{Without} \xrightarrow{\mathit{triv}} \mathsf{Interrupt} \tag{1}$$

$$\mathsf{Server} : \mathsf{NDChecking}_i \xrightarrow{\mathit{refuse}} \mathsf{Idle} * \mathsf{Client}_i(\mathsf{CS}) : \mathsf{Interrupt} \xrightarrow{\mathit{notYet}} \mathsf{Without} \tag{2}$$

$$\mathsf{Server} : \mathsf{NDChecking}_i \xrightarrow{\mathit{permit}} \mathsf{NDHelping}_i * \mathsf{Client}_i(\mathsf{CS}) : \mathsf{Interrupt} \xrightarrow{\mathit{request}} \mathsf{With} \tag{3}$$

$$\mathsf{Server} : \mathsf{NDHelping}_i \xrightarrow{\mathit{continue}} \mathsf{Idle} * \mathsf{Client}_i(\mathsf{CS}) : \mathsf{With} \xrightarrow{\mathit{done}} \mathsf{Without} \tag{4}$$

Note that for this protocol, each conductor step of the server corresponds to a phase change of exactly one client. E.g., the server moves from the state Idle to $\mathsf{NDChecking}_i$ iff the global client process $\mathsf{Client}_i(\mathsf{CS})$ changes from the phase Without to the phase Interrupt. The server then makes a $\mathtt{check}_i$ transition. In general, there is a precondition, however. Within the phase Without sufficient progress should have been made, such that the particular trap has been reached. In this case, it is the trivial trap triv rendering the requirement superfluous, as the trivial trap, containing all states of the phase Without, is trivially reached. For the actual checking, the next two consistency rules, dependent on the trap notYet and request, respectively, decide the target of the conductor transition and the next participant phase, viz. state Idle and phase Without or state $\mathsf{NDHelping}_i$ and phase With, respectively. The last consistency rule couples the conductor's returning from state $\mathsf{NDHelping}_i$ to Idle with trap done of phase With having been entered.

The consistency rules specify *horizontal* dynamic consistency, i.e. across components, here between server and clients. Such specification is about coordination, i.e. what Paradigm actually models, stepwise computation of next behavioural constraints. The constraining property imposed by a phase implies, an underlying $\mathsf{Client}_i$ transition is allowed only if it belongs to the phase that corresponds to the current state of the role of $\mathsf{Client}_i$ in the CS collaboration, i.e. the current state of the global STD $\mathsf{Client}_i(\mathsf{CS})$. The constraining property $\mathsf{Client}_i$ commits to by entering a trap, allows for a phase transfer, i.e. a transition of $\mathsf{Client}_i(\mathsf{CS})$, once the (connecting) trap is entered. These two constraining properties syntactically guarantee *vertical* dynamic consistency, i.e. within a component between its underlying STD and its role.

As mentioned in Section 1, is has become evident to us that separating detailed from global behaviour as supported by the Paradigm language, allows one to reason about reduction by abstraction in a rather natural way. The intuitive explanation for this is as follows: Global behaviour, actually defining phases a system needs to go through during a particular coordination solution, is built on top of the detailed behaviour: each global phase represents a sub-behaviour of the underlying detailed behaviour. Nevertheless, not every action at the detailed level affects the current global phase. Only some actions may enable a next phase transfer and hence may affect the protocol execution. Thus, it is natural to try to detect the detailed actions that do not matter for, i.e. that cannot be observed at, the protocol level. By hiding them, a reduced detailed behaviour is obtained, just containing all relevant information and actions needed for proper execution of the component role within the protocol. As we shall show for

our running example, this information can be extracted from the hierarchical structure per component in the Paradigm model, see Subsection 4.1. Note that all interaction between components (horizontal) and all hierarchical structure within components (vertical), as specified in the Paradigm model in an explicit manner, are flattened in the PA translation and hence their character being either horizontal or vertical, gets lost. Thus, after the PA translation only a single communication pattern remains, from which it is no longer straightforward to extract information needed for proper reduction of detailed behaviour.

Yet another aspect of the Paradigm model that can be justified and confirmed by the approach taken here is discussed shortly in the paper, see Subsection 4.2. From the definition of Paradigm, although provided with a formal operational semantics, it is not straightforward to see to what extent a component's detailed behaviour is not affected by some constraints or coordination rule. In particular, consistency rules for some complex model may have an unforeseen effect on detailed component behaviour, in particular a deadlock at the detailed level. The translation from Paradigm to ACP combined with the abstraction techniques discussed in the next section supports formal verification of separate protocols and of overall coordination.

# 3 Paradigm model as a process algebraic specification

In this section we show by means of the example introduced in Section 2, how a Paradigm model can be translated into ACP. The general translation has been defined in [3] to which we refer for more detail. Roughly, each STD will be represented by a recursive specification. Vertical consistency in Paradigm has to be expressed explicitly. In particular, to represent the interaction of a detailed STD and the global STD, we use actions $\mathtt{ok}!(.)$ and $\mathtt{ok}?(.)$ that take the labels of *detailed steps* as their argument. The complementary actions synchronize if the step of the detailed STD is allowed by the current phase of the global STD as constraint. Thus, synchronization of actions $\mathtt{ok}!(\cdot)$ and $\mathtt{ok}?(\cdot)$ between global STD and detailed STD reflect the current permission for the detailed step to be taken.

In addition, we use the complementary actions $\mathtt{at}!(.)$ and $\mathtt{at}?(.)$ that take *detailed states* as their arguments. The complementary actions synchronize if the step to be taken by the global STD is allowed by the current trap of the detailed STD as constraint. Upon synchronization of $\mathtt{at}!(\cdot)$ and $\mathtt{at}?(\cdot)$ the global process will update its trap information, if applicable. For the communication within the protocol, here between the server and its clients, actions $\mathtt{crule}!(.)$ on the side of a conductor are meant to complement $\mathtt{crule}?(.)$ actions on the side of the employees. Synchronization leads to execution of the corresponding consistency rule: a detailed transition of the conductor, phase changes for the employees involved.

For the concrete example the above amounts to the following. We adorn the $n$ processes $\mathsf{Client}_i$ with the actions $\mathtt{at}!$, conveying state information, and actions $\mathtt{ok}?$, regarding transition eligibility.

$$
\begin{aligned}
\widehat{\mathsf{Client}_i} &= \mathsf{Out}_i \\
\mathsf{Out}_i &= \mathtt{at}!(\mathsf{Out}_i)\cdot\mathsf{Out}_i + \mathtt{ok}?(\mathsf{enter}_i)\cdot\mathsf{Waiting}_i \\
\mathsf{Waiting}_i &= \mathtt{at}!(\mathsf{Waiting}_i)\cdot\mathsf{Waiting}_i + \mathtt{ok}?(\mathsf{explain}_i)\cdot\mathsf{Busy}_i \\
\mathsf{Busy}_i &= \mathtt{ok}?(\mathsf{thank}_i)\cdot\mathsf{AtDoor}_i \\
\mathsf{AtDoor}_i &= \mathtt{at}!(\mathsf{AtDoor}_i)\cdot\mathsf{AtDoor}_i + \mathtt{ok}?(\mathsf{leave}_i)\cdot\mathsf{Out}_i
\end{aligned}
$$

The LTS of $\widehat{\mathsf{Client}_i}$ of $\mathsf{Client}_i$ is given in Figure 4a (with the subscript $i$ suppressed). The definition of process $\widehat{\mathsf{Client}_i}$ assures, the process really starts in close correspondence to starting state $\mathsf{Out}$ from Figure 1a. The definition of process $\mathsf{Out}_i$ expresses: (1) upon being asked, it can exchange state information while keeping the process as-is; (2) it can ask for permission to take the analogue of transition $\mathsf{enter}$ from

Figure 1a, in view of continuing with process $\mathsf{Waiting}_i$ thereafter. Note, in the definition of process $\mathsf{Busy}_i$ the possibility for exchange of state information is not specified, as asking for it does never occur. Note, in Figure 1b, state Busy does not belong to trap done.

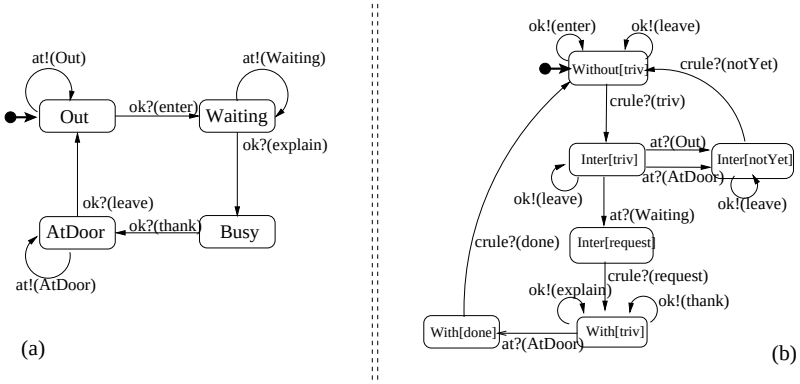


Figure 4: Processes (a) $\widehat{\mathsf{Client}}$ and (b) $\widehat{\mathsf{Client}}(\mathsf{CS})$.

In a similar manner, the $n$ processes $\mathsf{Client}_i(\mathsf{CS})$ are augmented with the actions at? and ok!. Now, at the global level, the relevant information is the pair of the current phase and the current trap. For example, the recursion variable $\mathsf{Without}_i[\mathsf{triv}]$ represents that $\mathsf{Client}_i$ is constrained to phase Without and hasn't reached a specific trap, whereas $\mathsf{Interrupt}_i[\mathsf{notYet}]$ reflects that $\mathsf{Client}_i$ committed to phase Interrupt resides in trap notYet. As these global processes play a participant role in the protocol, the crule? actions for engaging in a consistency rule have been put in place as well.

$$
\begin{aligned}
\widehat{\mathsf{Client}}_i(\mathsf{CS}) &= \mathsf{Without}_i[\mathsf{triv}] \\
\mathsf{Without}_i[\mathsf{triv}] &= \mathtt{ok!}(\mathsf{leave}_i)\cdot\mathsf{Without}_i[\mathsf{triv}] + \mathtt{ok!}(\mathsf{enter}_i)\cdot\mathsf{Without}_i[\mathsf{triv}] + \\
&\quad \mathtt{crule?}(\mathsf{triv}_i)\cdot\mathsf{Interrupt}_i[\mathsf{triv}] \\
\mathsf{Interrupt}_i[\mathsf{triv}] &= \mathtt{at?}(\mathsf{AtDoor}_i)\cdot\mathsf{Interrupt}_i[\mathsf{notYet}] + \mathtt{at?}(\mathsf{Out}_i)\cdot\mathsf{Interrupt}_i[\mathsf{notYet}] + \\
&\quad \mathtt{at?}(\mathsf{Waiting}_i)\cdot\mathsf{Interrupt}_i[\mathsf{request}] + \mathtt{ok!}(\mathsf{leave}_i)\cdot\mathsf{Interrupt}_i[\mathsf{triv}] \\
\mathsf{Interrupt}_i[\mathsf{notYet}] &= \mathtt{ok!}(\mathsf{leave}_i)\cdot\mathsf{Interrupt}_i[\mathsf{notYet}] + \mathtt{crule?}(\mathsf{notYet}_i)\cdot\mathsf{Without}_i[\mathsf{triv}] \\
\mathsf{Interrupt}_i[\mathsf{request}] &= \mathtt{crule?}(\mathsf{request}_i)\cdot\mathsf{With}_i[\mathsf{triv}] \\
\mathsf{With}_i[\mathsf{triv}] &= \mathtt{at?}(\mathsf{AtDoor}_i)\cdot\mathsf{With}_i[\mathsf{done}] + \mathtt{ok!}(\mathsf{explain}_i)\cdot\mathsf{With}_i[\mathsf{triv}] + \\
&\quad \mathtt{ok!}(\mathsf{thank}_i)\cdot\mathsf{With}_i[\mathsf{triv}] \\
\mathsf{With}_i[\mathsf{done}] &= \mathtt{crule?}(\mathsf{done}_i)\cdot\mathsf{Without}_i[\mathsf{triv}]
\end{aligned}
$$

The corresponding LTS of the specification $\widehat{\mathsf{Client}}_i(\mathsf{CS})$ of $\mathsf{Client}_i(\mathsf{CS})$ is given in Figure 4b.

As above, process $\widehat{\mathsf{Client}}_i(\mathsf{CS})$ is defined in close correspondence to $\mathsf{Without}_i[\mathsf{triv}]$ being starting state in Figure 2b. The ok!(.)-actions provide the permission answers to requests from $\widehat{\mathsf{Client}}_i$ to take a detailed step. The at?(.)-actions ask for state information relevant for deciding a next, smaller trap has been entered. The crule?(.)-actions correspond to a phase change, so they synchronize with a particular conductor step.

The final component of the Paradigm model that needs to be translated into ACP is the non-deterministic server Server. In fact, the STD of the server as given in Figure 3 exactly corresponds to its recursive specification; we only rename each transition label $\ell$ from Figure 3 into $\mathtt{crule!}(\ell)$ to stay consistent with the general translation as defined in [3], for instance $\mathsf{permit}_i$ is renamed into $\mathtt{crule!}(\mathsf{permit}_i)$ in the PA

specification. There is neither any $\mathtt{ok}(.)$ action nor any $\mathtt{at}(.)$ action added here. This component plays the conductor role in the protocol and as such it is represented only by its detailed behaviour (detailed STD). Therefore, no vertical constraints are imposed on its detailed behaviour.

$$\begin{aligned}
\widehat{\mathsf{Server}} &= \mathsf{Idle}\\
\mathsf{Idle} &= \mathtt{crule!}(\mathsf{check}_1)\cdot\mathsf{NDChecking}_1 + \cdots + \mathtt{crule!}(\mathsf{check}_n)\cdot\mathsf{NDChecking}_n\\
\mathsf{NDChecking}_i &= \mathtt{crule!}(\mathsf{permit}_i)\cdot\mathsf{NDHelping}_i + \mathtt{crule!}(\mathsf{refuse}_i)\cdot\mathsf{Idle}\\
\mathsf{NDHelping}_i &= \mathtt{crule!}(\mathsf{continue}_i)\cdot\mathsf{Idle}
\end{aligned}$$

For the communication function '|' we put $\mathtt{at!}(s)\,|\,\mathtt{at?}(s) = \tau$ for 'states' $s = \mathsf{Out}_i, \mathsf{Waiting}_i, \mathsf{AtDoor}_i$, and $\mathtt{ok?}(a)\,|\,\mathtt{ok!}(a) = \mathtt{ok}(a)$, for actions $a = \mathsf{enter}_i, \mathsf{explain}_i, \mathsf{thank}_i, \mathsf{leave}_i$. Note, ACP allows to keep the result of the synchronization of $\mathtt{ok?}(a)$ and $\mathtt{ok!}(a)$ observable, here as the action $\mathtt{ok}(a)$, for suitable $a$. We exploit this feature below to express system properties, since the synchronization actions $\mathtt{ok}(a)$ describe detailed steps taken by clients. E.g., observing $\mathtt{ok}(\mathsf{enter}_i)$ indicates a service request made by $\mathsf{Client}_i$. On the contrary, synchronization of $\mathtt{at!}()$ and $\mathtt{at?}()$ is only used to update the information of the current detailed state. The resulting actions are internal to the component and not needed in any further analysis. Therefore, we safely use $\tau$ for the synchronization of $\mathtt{at?}()$ and $\mathtt{at!}()$.

Finally, we need to encode the coordination captured by the consistency rules. For example, consistency rule (1) couples a detailed $\mathsf{check}_i$ step of the $\mathsf{Server}$, being the conductor of the $\mathsf{CS}$ protocol, to the global $\mathsf{triv}$ step of $\mathsf{Client}_i$, being a participant in the $\mathsf{CS}$ protocol. The net result is a state transfer, i.e. a transition $\mathsf{Idle} \xrightarrow{\mathit{check}_i} \mathsf{NDChecking}_i$ for the server, and a phase transfer, i.e. a transition $\mathsf{Without} \xrightarrow{\mathit{triv}} \mathsf{Interrupt}$ in the global STD for the $i$-th client. Similar correspondences apply to the other consistency rules. Therefore, we put

$$\begin{aligned}
\mathtt{crule!}(\mathsf{check}_i)\,|\,\mathtt{crule?}(\mathsf{triv}_i) &= \mathsf{check}_i & \mathtt{crule!}(\mathsf{refuse}_i)\,|\,\mathtt{crule?}(\mathsf{notYet}_i) &= \mathsf{refuse}_i\\
\mathtt{crule!}(\mathsf{permit}_i)\,|\,\mathtt{crule?}(\mathsf{request}_i) &= \mathsf{permit}_i & \mathtt{crule!}(\mathsf{continue}_i)\,|\,\mathtt{crule?}(\mathsf{done}_i) &= \mathsf{continue}_i
\end{aligned}$$

As usual, unmatched synchronization actions will be blocked to enforce communication. We collect those in the set $A = \{\,\mathtt{crule!}, \mathtt{crule?}, \mathtt{at?}, \mathtt{at!}, \mathtt{ok?}, \mathtt{ok!}\,\}$. Finally, the process for the collaboration of the server and the $n$ clients is given by

$$\partial_A(\widehat{\mathsf{Client}}_1 \,\|\, \widehat{\mathsf{Client}}_1(\mathsf{CS}) \,\|\, \ldots \,\|\, \widehat{\mathsf{Client}}_n \,\|\, \widehat{\mathsf{Client}}_n(\mathsf{CS}) \,\|\, \widehat{\mathsf{Server}}) \tag{5}$$

The next section is concerned with the intertwining of detailed and the global behavior, and possible ways to reduce the component specification by abstracting away from specific detailed activities. The process algebraic specification of our running client-server example will be used below to establish relations between Paradigm models before and after reduction. Therefore, it comes in handy to represent the overall behaviour of the $\mathsf{Client}$ component as the parallel composition of its detailed and global behaviour. To this end, we denote the set of states of the detailed process $\widehat{\mathsf{Client}}$ by $\mathsf{States}_D = \{\,\mathsf{Out}, \mathsf{Waiting}, \mathsf{Busy}, \mathsf{AtDoor}\,\}$, the set of labels of its transitions by of detailed $\mathsf{Labels}_D = \{\,\mathsf{enter}, \mathsf{explain}, \mathsf{thank}, \mathsf{leave}\,\}$ and we put

$$\mathsf{AT} = \{\,\mathtt{at!}(s), \mathtt{at?}(s) \mid s \in \mathsf{States}_D\,\} \qquad \mathsf{OK} = \{\,\mathtt{ok!}(a), \mathtt{ok?}(a) \mid a \in \mathsf{Labels}_D\,\}$$

and define $H = \mathsf{AT} \cup \mathsf{OK}$. Then the process combining detailed behaviour of $\widehat{\mathsf{Client}}$ and global behaviour of $\widehat{\mathsf{Client}}(\mathsf{CS})$ can be expressed as $\widehat{\mathsf{Client}}(\mathsf{DG})$, with $\mathsf{DG}$ referring to 'detailed' and 'global', given by

$$\widehat{\mathsf{Client}}(\mathsf{DG}) = \partial_H(\widehat{\mathsf{Client}} \,\|\, \widehat{\mathsf{Client}}(\mathsf{CS}))$$

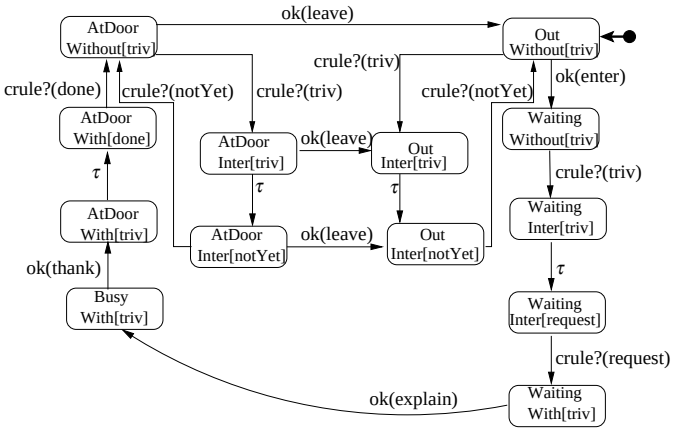


Figure 5: Process $\widehat{\mathsf{Client}}(\mathsf{DG})$

Figure 5 shows the behavior of $\widehat{\mathsf{Client}}(\mathsf{DG})$ graphically. The process describes the way the detailed and global behaviors occur and constrain each other.

On the one hand, steps taken at the detailed level influence the current phase at the global level, and therefore allows and forbids certain phase transitions at the global level. The global process and its transitions, are 'navigated' by the activities executed at the detailed level. For instance, the effect of the detailed transition $\mathtt{ok}(\mathsf{enter})$ is described with the appearance of two $\mathsf{triv}$ transitions. One of them captures the scenario in which the client has not yet required any service, which means that $\mathsf{enter}$ has not been taken yet at the detailed level, although the server (conductor) may offer service. It can be observed that this transition is followed by the phase transition $\mathsf{notYet}$ which brings the process back to the initial state. We can also observe that as soon as the detailed transition $\mathsf{enter}$ is taken, the enabled $\mathsf{triv}$ transition differs from the previous one.

On the other hand, from $\widehat{\mathsf{Client}}(\mathsf{DG})$ we can observe how each phase, i.e. a global state, constrains the steps that can be taken locally. Moreover, it is specified exactly how a trap that is reached blocks any detailed transitions, just as expected. For instance, we see that the action $\mathtt{ok}(\mathsf{leave})$ on top of Figure 5 cannot be executed before the phase is changed, i.e. a step from $\mathsf{With}[\mathsf{done}]$ to $\mathsf{Without}[\mathsf{triv}]$ via the global transition $\mathtt{crule?}(\mathsf{done})$. Note that such details, which are explicit and easily observable from the ACP specification of the composition $\widehat{\mathsf{Client}}(\mathsf{DG})$, cannot be directly detected in the Paradigm model.

Once systems are modeled algebraically, their behaviours can be compared. Comparison is typically done by means of equivalence relations, chosen appropriately to preserve certain properties. Since we aim at the mCRL2 toolset for tool support, we choose for branching bisimulation [8] as the equivalence relation we apply. Indeed, branching bisimulation is the strongest in the spectrum of behavioural equivalence relations, but yet weak enough to identify sufficiently many systems. Below we adapt the definition from [8] (originally defined on labelled transition systems) to STDs with uniquely indicated initial states. In fact, labelled transition systems (LTS), as a (visual) representation of process algebraic specifications, can be seen also as STDs. Therefore, in the sequel we do not make explicit distinction between LTSs and STDs.

**Definition 1.** *For two STDs $Z = \langle \mathtt{ST}, \mathtt{AC}, \mathtt{TS} \rangle$, $Z' = \langle \mathtt{ST}', \mathtt{AC}', \mathtt{TS}' \rangle$ a symmetric relation $R \subseteq \mathtt{ST} \times \mathtt{ST}'$ is called a branching bisimulation relation if for all $s \in \mathtt{ST}$ and $t \in \mathtt{ST}'$ such that $R(s,t)$, the following condition is met: if $s \xrightarrow{a} s'$ in $Z$, for some $a \in \mathtt{AC} \cup \{\tau\}$, then either $a = \tau$ and $R(s',t)$, or for some $n \geq 0$, there exist $t_1, \ldots, t_n$ and $t'$ in $\mathtt{ST}'$ such that $t \xrightarrow{\tau} t_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} t_n \xrightarrow{a} t'$ in $Z'$, $R(s,t_1), \ldots, R(s,t_n)$ and $R(s',t')$.*

For two STDs $Z$ and $Z'$, two states $s \in Z$ and $t \in Z'$ are called branching bisimilar, notation $s \leftrightarrows_b t$, if there exists a branching bisimulation relation $R$ for $Z$ and $Z'$ such that $R(s,t)$. The STDs $Z$ and $Z'$ are branching bisimilar, notation $Z \leftrightarrows_b Z'$ if their initial states are branching bisimilar.

# 4 Reduction of the client processes

In Section 3 we explained how ACP specifications are obtained from the detailed and global client STDs, and how ACP's communication function captures synchronization of detailed and global steps, guaranteeing consistent dynamics at both levels. Based on the complete client component we are able to make several observations regarding the Paradigm approach to separate the detailed from the global behaviour.

## 4.1 First-reduce then-compose

The global STD of a component is an abstract representation of its detailed STD. It represents the part of the behaviour of the component that is essential for the interaction within a given collaboration. In general, for the global behaviour not all local transitions are relevant, most are not influencing the overall coordination at all. Although not always easy to isolate, in actual full-fledged systems only a restricted part of the whole system provides a specific functionality. In such a situation, from a modeling perspective it is clarifying to abstract away the irrelevant part and to concentrate on a reduced detailed behaviour containing the relevant interaction. As a consequence, dealing with models that are purposely made concise becomes simpler, more feasible and less error-prone.

In the previous sections, we have made a Paradigm model out of the components: detailed client STDs, their global STDs and the server STD. Moreover, we have presented their translations into process algebraic specifications. The overall behaviour of the client-server system is obtained by putting the components involved in parallel and make them interact. In this section we show that we can achieve the same total behaviour of the client-server system by *first reducing* the client components and *then composing* the reduced versions afterwards with other components of the system. Reduction is directly applied on the original Paradigm client model, by abstracting away irrelevant states and local transitions.

It is intuitively clear that the global behaviour alone is not branching bisimilar to the overall client behaviour $\widehat{\mathsf{Client}}(\mathsf{DG})$. This is because some local steps change the further global behaviour. As a consequence, such local transitions can be detected at the global level. Extending terminology going back to [8], we call these transitions *globally non-inert*. Similarly, a local transition is referred to as *globally inert* if it cannot be observed, explicitly or implicitly, at the global level. More specifically, it can be detected whether local action enter has been taken or not by observing whether the global transition notYet or global transition request follows after global step triv. Putting it differently, the transition labeled enter makes the difference for phase Interrupt of residing in trap notYet or in trap request, as can be seen in Figure 1. Thus, the local transition enter is not globally inert. In a similar manner, the local action thank is not globally inert as it enables –and so it can be detected– the execution of the global action done. In terms of the partition, in phase With the action thank enters the trap done. On the other hand, again referring to the phases of Client(CS) in Figure 1b, we see that the action leave is in each phase either within a trap (phases Without and Interrupt) or not possible at all (phase With is missing the target state Out). Likewise, the action explain is not possible (phases Without and Interrupt are missing state Busy) or doesn't change the trap information (in phase With the transition doesn't enter the trap done).

**Definition 2.** *Let a Paradigm model be given. A detailed transition $x \xrightarrow{a} x'$ of a participant of a protocol is called* globally inert *with respect to its partition $\pi = \{ (S_i, T_i) \mid i \in I \}$ if for all traps $t$ in $T_i$ it holds that $x \in t \iff x' \in t$ whenever both $x, x' \in S_i$,*

*$i \in I$. An action $a$ is called globally inert for a participant of a protocol with respect to a partition, if all $a$-labeled transitions are.*

Using the notion of detailed transitions being globally inert or non-inert, we can reduce the detailed STD of the client. After renaming all globally inert transitions into $\tau$, we can identify branching bisimilar states. The resulting quotient STD for the client carries the behaviour that is necessary and sufficient for the global STD to interact with the other components, including the conductor of the collaboration. The composition of the process algebraic specifications of the quotient STD and the global $\widehat{\mathsf{Client}}(\mathsf{CS})$ behaves exactly (up to branching bisimulation) as the behaviour of the composition of the original detailed and global STDs together as represented by $\widehat{\mathsf{Client}}(\mathsf{DG})$. By congruence, composition of either of these systems with the other clients and the server leads, modulo branching bisimulation equivalence, to the same behaviour. This is summarized by the next result, where $\tau_I$, for a set of labels $I$, represents the hiding of the actions in $I$ from $P$ by renaming them into $\tau$, and $\partial_J(P)$, for a set of labels $J$, is the encapsulation of the actions of $J$ from $P$ by blocking and transition for $P$ with label in $J$.

**Lemma 3.** *Let $G \subseteq \mathit{Labels}_D$ be a subset of globally inert actions. Then it holds for the induced quotient QClient of Client that*

*(i)* $\mathit{QClient} \leftrightarrows_b \tau_G(\mathit{Client})$, *and*

*(ii)* $\partial_H(\,\widehat{\mathit{QClient}} \,\|\, \widehat{\mathit{Client}}(\mathit{CS})\,) \leftrightarrows_b \tau_{\mathit{OK}(G)}(\,\widehat{\mathit{Client}}(\mathit{DG})\,)$, *where* $\mathit{OK}(G) = \{\, \mathtt{ok}(a) \mid a \in G \,\}$.

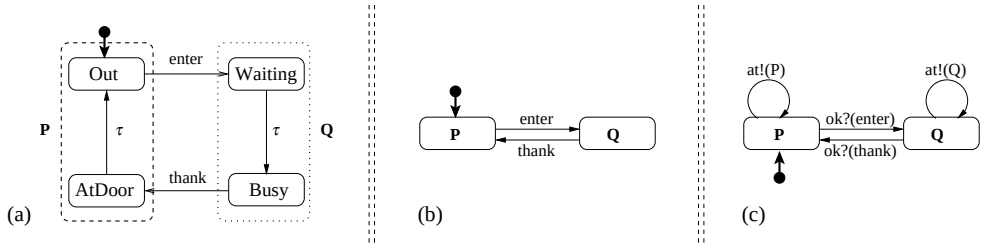


Figure 6: (a) process $\tau_G(\mathsf{Client})$ and related states, (b) quotient STD QClient and (c) $\widehat{\mathsf{QClient}}$.

*Proof.* We consider the case of the maximal set of local actions that are globally inert, i.e. for $G = \{\, \mathsf{explain}, \mathsf{leave} \,\}$. Split the set of states $\mathsf{States}_D$ of the detailed STD into $P = \{\, \mathsf{Out}, \mathsf{AtDoor} \,\}$ and $Q = \{\, \mathsf{Waiting}, \mathsf{Busy} \,\}$. Let QClient be the induced quotient STD, the STD obtained from Client by identifying the states Out and AtDoor as well as the states Waiting and Busy. The processes QClient and $\tau_G(\mathsf{Client})$ are shown in Figure 6ab. A branching bisimulation between QClient and $\tau_G(\mathsf{Client})$ can be immediately established, which proves the first part of the lemma.

In order to prove the second part of the lemma, we first translate QClient into the process algebraic specification $\widehat{\mathsf{QClient}}$ whose STD is shown in Figure 6c. In order to compute the composition of $\widehat{\mathsf{QClient}}$ and $\widehat{\mathsf{Client}}(\mathsf{CS})$ the communication function has to be adapted to $\widehat{\mathsf{QClient}}$. For the $\widehat{\mathsf{QClient}}$ process Out and AtDoor are identified into the $P$. Similar for Waiting, Busy, now represented by $Q$. Thus, a detailed $\widehat{\mathsf{QClient}}$ communication intention conveying '*at P*' or '*at Q*' updates the global process about the current local state. Hence, we extend the communication function with $\mathtt{at!}(P) \,|\, \mathtt{at?}(\mathsf{Out}) = \tau$, $\mathtt{at!}(P) \,|\, \mathtt{at?}(\mathsf{AtDoor}) = \tau$, $\mathtt{at!}(Q) \,|\, \mathtt{at?}(\mathsf{Waiting}) = \tau$ and $\mathtt{at!}(Q) \,|\, \mathtt{at?}(\mathsf{Busy}) = \tau$. Now we consider the process $\partial_H(\,\widehat{\mathsf{QClient}} \,\|\, \widehat{\mathsf{Client}}(\mathsf{CS})\,)$ with $H = \mathsf{AT} \cup \mathsf{OK}$ as defined in Section 3, with AT extended accordingly. The composition is shown in Figure 7a, the process $\tau_{\mathsf{OK}(G)}(\,\widehat{\mathsf{Client}}(\mathsf{DG})\,)$ is depicted in Figure 7b. It is straightforward to establish a branching bisimulation between these two processes. □

State names of $\tau_{\mathsf{OK}(G)}(\,\widehat{\mathsf{Client}}(\mathsf{DG})\,)$ have been suppressed in Figure 7b for readability. Note that the number of states in $\tau_G(\,\widehat{\mathsf{Client}}(\mathsf{DG})\,)$ is 13, while the *first-reduce then-compose* approach with $\widehat{\mathsf{QClient}}$ and $\widehat{\mathsf{Client}}(\mathsf{CS})$ generates a process with 9 states only. See table 1 below for more numerical results.

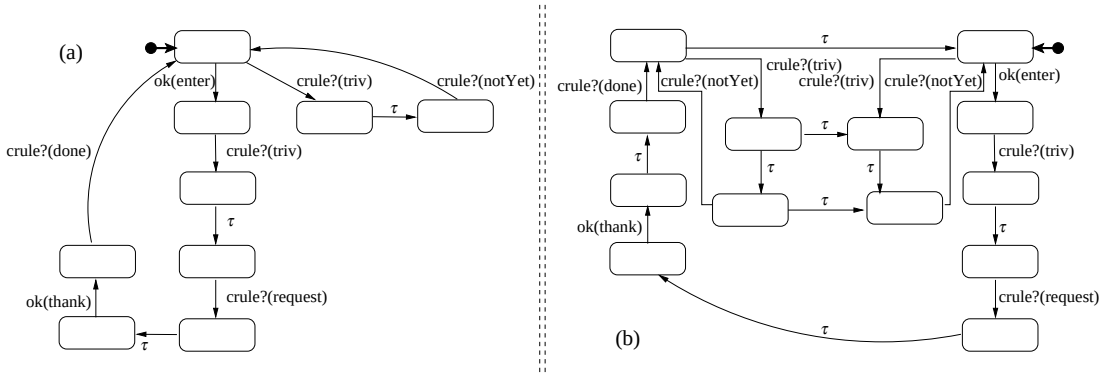


Figure 7: Branching bisimilar processes: (a) $\partial_H(\widehat{\mathsf{QClient}} \,\|\, \widehat{\mathsf{Client}}(\mathsf{CS}))$ (b) process $\tau_{\mathsf{OK}(G)}(\widehat{\mathsf{Client}}(\mathsf{DG}))$.

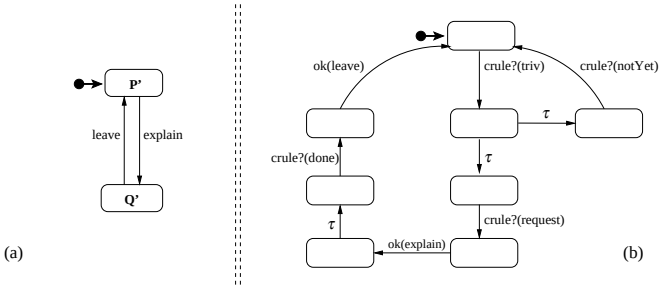


Figure 8: (a) adapted quotient process QClient, (b) composition of new $\widehat{\mathsf{QClient}}$ and $\widehat{\mathsf{Client}}(\mathsf{CS})$.

It is obvious that not every choice of actions at the detailed level has the property of Lemma 3. For example, selecting the set of actions $G' = \{\mathsf{enter}, \mathsf{thank}\}$, yields a split-up into $\{\mathsf{Out}, \mathsf{Waiting}\}$ and $\{\mathsf{Busy}, \mathsf{AtDoor}\}$ and another reduction, depicted in Figure 8a. However, this reduction is not a proper one as the induced composition of the reduced detailed and the global behaviour in Figure 8 is not branching bisimilar with the original composition $\tau_{\mathsf{OK}(G')}(\widehat{\mathsf{Client}}(\mathsf{DG}))$.

It is instructive to consider a slightly different client. Now we assume that the client may decide to draw back the service request and return back to the initial state Out. The detailed STD and the global STD shown in Figure 9 differ from the model in Figure 1 only in the return transition. If we apply the same reasoning of Lemma 3 to this model of a client, we observe that the return transition does not change the situation regarding the reduction of the local behaviour. Again, the enter transition is not globally inert, for the same reasons as in the previous model. Similarly, return is also not globally inert. Still, the original quotient from Lemma 3 based on the inert actions explain and leave yields a proper reduction. See Figure 10.

The last example we consider as a further variation, named $\mathsf{Client}''$, is presented in Figure 11. The only change is now in the global STD $\mathsf{Client}''(\mathsf{CS})$. The client is provided service unconditionally, i.e. without interruption, even without needing it. But, if it doesn't need it the client is handled as if it does not need service *any longer*. The simplified global behaviour, with less phases and less traps, imposes less constraints on the detailed behaviour. Thus, the relation between the detailed and the global behaviour

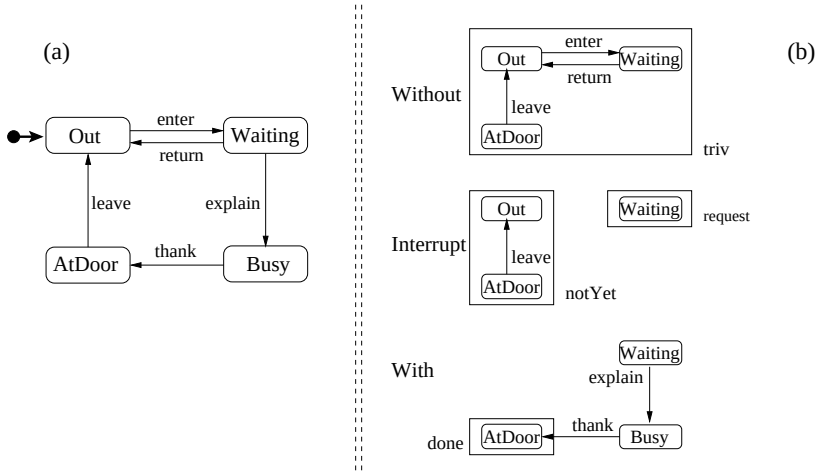


Figure 9: Modified client: (a) STD of $\mathsf{Client}'$, (b) phase and trap constraints.

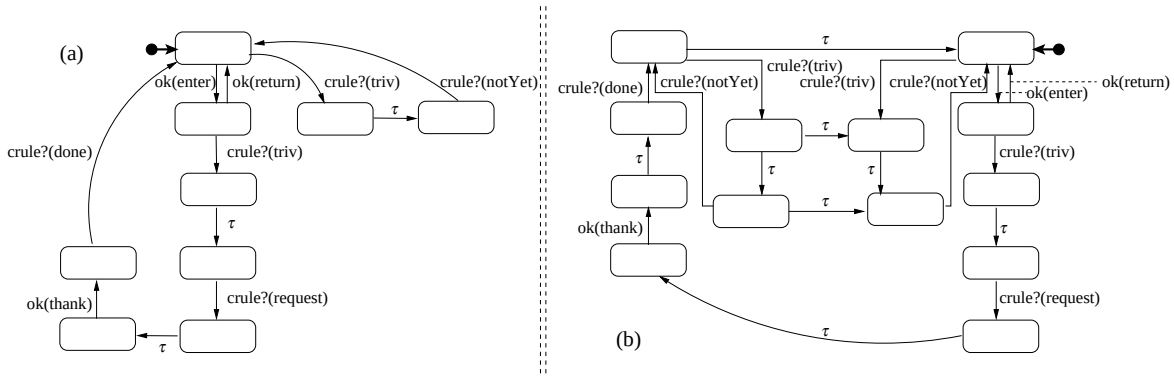


Figure 10: Branching bisimilar processes: (a) $\partial_H(\mathsf{QClient}' \,\|\, \widehat{\mathsf{Client}}'(\mathsf{CS}))$, (b) $\tau_{\mathsf{OK}(G)}(\widehat{\mathsf{Client}}'(\mathsf{DG}))$.

is rather loose. In Figure 12 the behaviour of process $\widehat{\mathsf{Client}}''(\mathsf{CS})$ and the parallel composition $\widehat{\mathsf{Client}}''(\mathsf{DG})$ are graphically represented. In order to show this formally, we again apply the *first-reduce then-compose* approach along the lines of Lemma 3 by taking the trivial split-up of $\mathsf{States}_D$ along all detailed actions in $\mathsf{Labels}_D$. Thus, we identify all local actions in $G'' = \mathsf{Labels}_D$ as globally inert. The resulting quotient STD of $\mathsf{QClient}''$ and its process algebraic translation are shown in Figure 13bc. The composition of the reduced detailed behaviour of $\mathsf{Client}''$ with its global behaviour has now 3 states as shown in Figure 13d. A branching bisimulation between this process and the corresponding process $\tau_{G''}(\widehat{\mathsf{Client}}''(\mathsf{DG}))$ can be established easily.

In order to investigate the effect of the reduction on a larger scale, we have analyzed the client-server system using the mCRL2 toolset [11] and compared the implementation of the system using either the original $\widehat{\mathsf{Client}}$ components or their reduced versions $\widehat{\mathsf{QClient}}$. The translation of ACP-based specifications of the $n$ clients $\widehat{\mathsf{Client}}_i$, the global $\widehat{\mathsf{Client}}_i(\mathsf{CS})$ and the server $\mathsf{Server}$ into the input language of the mCRL2 toolset, which we use for our model analysis, is largely straightforward (see also [3]). Indeed, the application of

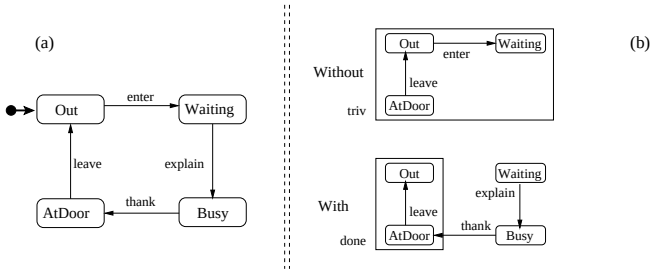


Figure 11: The Paradigm model of Client″.

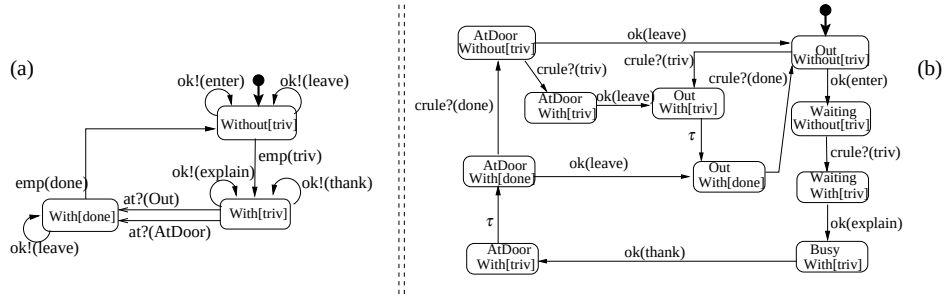


Figure 12: Processes $\widehat{\mathsf{Client}}''(\mathsf{CS})$ and $\widehat{\mathsf{Client}}''(\mathsf{DG})$.

the *first-reduce then-compose* principle yields a significant decrease in the size of the state space in a number of cases. The results are collected in Table 1.

## 4.2 Extracting detailed behaviour

Intuitively it is clear that in the case of the client-server example the global behaviour does not change or influence the local behaviour. In fact, if in the total client behaviour $\widehat{\mathsf{Client}}(\mathsf{DG})$ we hide the actions $\mathtt{crule?}(\cdot)$ from the set $E$ performed by the global process ($E$ for external), we obtain a process which is branching bisimilar to the detailed behaviour Client. This is expressed by the following lemma.

**Lemma 4.** *Client* $\leftrightarrows_b \tau_E(\widehat{\mathsf{Client}}(\mathsf{DG}))$.

*Proof.* We start from the process $\widehat{\mathsf{Client}}(\mathsf{DG})$ as shown in Figure 5. After hiding the actions in $E$, i.e. renaming them into $\tau$, the process $\tau_E(\widehat{\mathsf{Client}}(\mathsf{DG}))$ is obtained, shown in Figure 14. A branching bisimulation equivalence between this process and

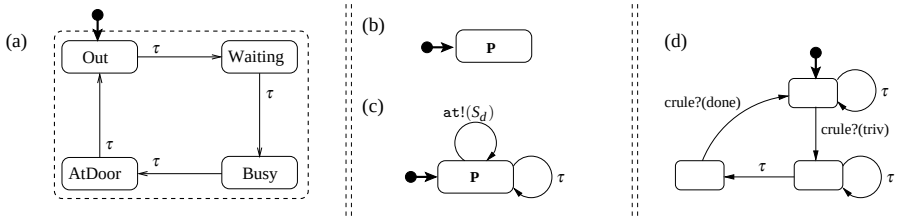


Figure 13: (a) $\tau_{G''}(\mathsf{Client}'')$, (b) $\widehat{\mathsf{Client}}''$, (c) $\widehat{\mathsf{QClient}}''$, (d) composition of $\widehat{\mathsf{QClient}}''$ and $\widehat{\mathsf{Client}}''(\mathsf{CS})$.

| n | with $\widehat{\mathsf{Client}}$ | | with $\widehat{\mathsf{QClient}}$ | |
|---|---|---|---|---|
| | states | transitions | states | transitions |
| 2 | 69 | 142 | 32 | 54 |
| 3 | 297 | 819 | 92 | 204 |
| 4 | 1161 | 3996 | 240 | 656 |
| 5 | 4293 | 17685 | 592 | 1920 |
| 6 | 15309 | 73386 | 1408 | 5280 |
| 10 | – | – | 36863 | 212480 |

(no result for $\widehat{\mathsf{Client}}$ with $n$=10 within 24 hours)

Table 1: Effect of the *first-reduce then-compose* approach.

Client process can be defined without difficulty. In Figure 15 related states are connected by differently dotted lines. Note, we have mirrored the Client orientation with respect to the North-East South-West diagonal. □

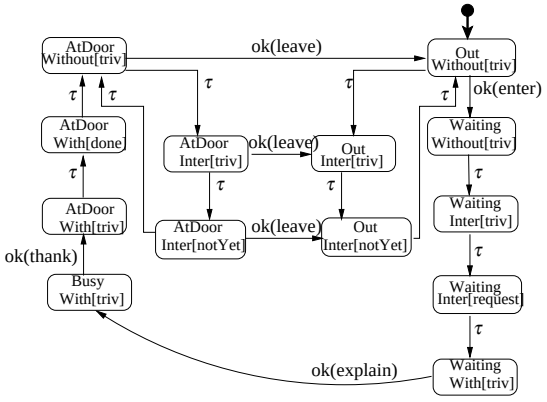


Figure 14: Process $\tau_E(\widehat{\mathsf{Client}}(\mathsf{DG}))$.

In the general situation, the statement of the lemma provides a check on the constraints imposed by the global STD on the detailed one. In case the statement of the lemma holds, the complete behaviour of the component is preserved in the consistent composition, assuming the coordinating protocol provides all phase transfers in some order. In case the statement of the lemma does not hold, part of the original detailed behaviour has been eliminated because of the participation with the protocol. This may be deliberate and allows for further reduction of the detailed STD. This may be accidental, requiring the overall coordination to be revised.

## 5 Concluding remarks

In a Paradigm model several STDs may belong to the same component, describing the component's dynamics either at various levels of abstraction (detailed vs. global STDs) or describing different roles

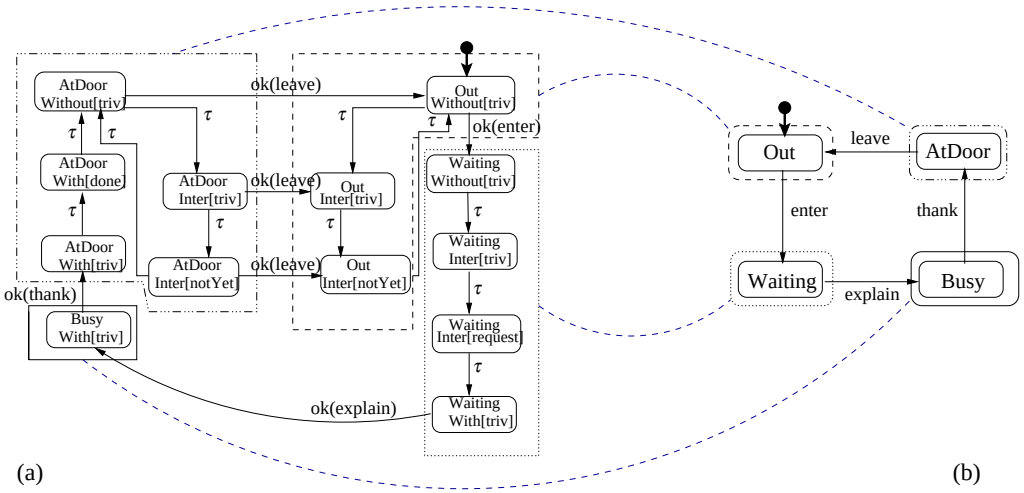


Figure 15: Branching bisimulation between (a) $\tau_E(\widehat{\mathsf{Client}}(\mathsf{DG}))$ and (b) Client.

of the component in various collaborations. Collaboration between components is described in terms of dynamic constraints. Vertical consistency is maintained by keeping phases vs. detailed transitions and traps vs. transfers aligned. Starting point of our investigation here is the translation of Paradigm models into the process algebra ACP and its coupling with the `mCRL2` toolset for subsequent automated analysis. In the translated model, every STD from the Paradigm model is represented by a recursive specification; the total behaviour of a single component is obtained as a composition of the recursive specifications of the detailed and the global component's STDs; the overall system is specified as a parallel composition of all components.

In this paper we have described a method to reduce the Paradigm representation of the detailed STDs of the components, yielding reduction of the overall Paradigm models, but preserving the overall behaviour. The reduction boils down to inferring globally inert detailed steps. By abstracting them away a smaller representation of the detailed component is obtained. This representation contains all information about the constraints the detailed behaviour imposes on the global behaviour(s) of the component. The formal validation that the reduction, indeed, does not change the overall model behaviour is achieved via the process algebraic representation of the model: we show for our client-server example that the reduced model is branching bisimilar to the original one, having the same properties. Furthermore, by means of a proper abstraction, in this case applied at the global level, we can observe directly from the model, by a direct comparison, in which way the global behaviour, and thus the collaboration, affects the components' detailed behaviour. In case no influence is to be expected, it is sufficient to show that the component model is equivalent, up to branching bisimulation, to the detailed behaviour after all global steps are abstracted away.

As to the contribution of this paper, we have established a further connection of process algebra and its supporting apparatus to the domain of coordination. In particular, abstraction and equivalences, typical for process algebra, become techniques that can be applied to coordination models, via the established link of the Paradigm language and ACP, in our case. Thus, coordination can be initially modeled in the Paradigm language which offers compositional and hierarchical modeling flexibility. Then, model reduction can be applied, if appropriate. Finally, via its process representation the model can be formally analyzed.

As future work we want to address the reduction of general Paradigm models and property guided

reduction, in particular in a situation with overlapping or orthogonal coordination. More specifically, it is interesting to study the notion of globally inert detailed steps for a component that participates in multiple collaborations. We plan to investigate whether other techniques from process algebraic analysis, e.g. iterated abstraction, and pattern-based simplifications can be beneficial for the modeling with Paradigm.

## References


[1] R.J. Allen (1997): *A Formal Approach to Software Architectures*. Ph.D. thesis, Carnegie Mellon University.

[2] S. Andova, L.P.J. Groenewegen, J.H.S. Verschuren & E.P. de Vink (2009): *Architecting Security with Paradigm*. In: *Architecting Dependable Systems VI*. LNCS 5835, pp. 255–283. doi:10.1007/978-3-642-10248-6_11.

[3] S. Andova, L.P.J. Groenewegen & E.P. de Vink (2010): *Dynamic Consistency in Process Algebra: From Paradigm to ACP*. *Science of Computer Programming* doi:10.1016/j.scico.2010.04.011, 45pp.

[4] S. Andova, L.P.J. Groenewegen & E.P. de Vink (2010): *Towards Dynamic Adaptation of Probabilistic Systems*. In: T. Margaria & B. Steffen, editors: *Proc. ISOLA 2010*. LNCS 6416, pp. 143–159. doi:10.1007/978-3-642-16561-0_19.

[5] F. Arbab (2004): *Reo: A Channel-based Coordination Model for Component Composition*. *Mathematical Structures in Computer Science* 14, pp. 329–366. doi:10.1017/S0960129504004153.

[6] J.C.M. Baeten, T. Basten & M.A. Reniers (2010): *Process Algebra: Equational Theories of Communicating Processes*, *Cambridge Tracts in Theoretical Computer Science* 50. CUP.

[7] G. Engels, R. Heckel, J.M. Küster & L. Groenewegen (2002): *Consistency-Preserving Model Evolution through Transformations*. In: J.-M. Jézéquel, H. Hußmann & S. Cook, editors: *Proc. UML 2002*. LNCS 2460, pp. 212–226. doi:10.1007/3-540-45800-X_18.

[8] R.J. van Glabbeek & P. Weijland (1996): *Branching time and abstraction in bisimulation semantics*. *Journal of the ACM* 43, pp. 555–600. doi:10.1145/233551.233556.

[9] L. Groenewegen & E. de Vink (2002): *Operational Semantics for Coordination in Paradigm*. In: F. Arbab & C. Talcott, editors: *Proc. Coordination 2002*. LNCS 2315, pp. 191–206. doi:10.1007/3-540-46000-4_20.

[10] L. Groenewegen & E. de Vink (2006): *Evolution-On-The-Fly with Paradigm*. In: P. Ciancarini & H. Wiklicky, editors: *Proc. Coordination 2006*. LNCS 4038, pp. 97–112. doi:10.1007/11767954_7.

[11] J.F. Groote et al. (2007): *The Formal Specification Language mCRL2*. In: E. Brinksma et al., editor: *Methods for Modelling Software Systems*. IBFI, Schloss Dagstuhl. 34 pages.

[12] N. Kokash, C. Koehler & E.P. de Vink (2010): *Time and Data-Aware Analysis of Graphical Service Models in Reo*. In: J.L. Fiadeiro A. Maggiolo-Schettini & S. Gnesi, editors: *Proc. SEFM 2010*. IEEE, pp. 125–134. doi:10.1109/SEFM.2010.26.

[13] C. Krause (2011): *Reconfigurable Component Connectors*. Ph.D. thesis, Leiden University. To appear.

[14] J.M. Küster (2004): *Consistency Management of Object-Oriented Behavioral Models*. Ph.D. thesis, University of Paderborn.

[15] M.Z. Kwiatkowska, G. Norman & D. Parker (2009): *PRISM: probabilistic model checking for performance and reliability analysis*. *SIGMETRICS Performance Evaluation Review* 36, pp. 40–45. doi:10.1145/1530873.1530882.

[16] M. Möller, E.-R. Olderog, H. Rasch & H. Wehrheim (2008): *Integrating a formal method into a software engineering process with UML and Java*. *Formal Aspects of Computing* , pp. 161–204 doi:10.1007/s00165-007-0042-7.

[17] M.A. Pérez-Toledano, A. Navasa, J.M. Murillo & C. Canal (2007): *TITAN: a Framework for Aspect-Oriented System Evolution*. In: *Proc. ICSEA 2007, Cap Esterel*. IEEE. doi:10.1109/ICSEA.2007.77, 8 pages.

[18] N.F. Rodrigues & L.S. Barbosa (2005): *Architectural Prototyping: From CCS to .Net*. In: A. Mota & A.V. Moura, editors: *Proc. SBMF 2004*. ENTCS 130, pp. 151–167. doi:10.1016/j.entcs.2005.03.009.

[19] M. Sirjani, A. Movaghar, A. Shali & F.S. de Boer (2004): *Modeling and Verification of Reactive Systems using Rebeca*. *Fundamenta Informaticae* 63, pp. 385–410. doi:10.1.1.107.2074.