Efficient Compensation Handling via Subjective Updates

Jovana Dedeić
University of Novi Sad
Trg Dositeja Obradovića 6
21000
Novi Sad, Serbia
radenovicj@uns.ac.rs

Jovanka Pantović
University of Novi Sad
Trg Dositeja Obradovića 6
21000
Novi Sad, Serbia
pantovic@uns.ac.rs

Jorge A. Pérez
University of Groningen &
CWI, Amsterdam
Nijenborgh 9, 9747AG
The Netherlands
j.a.perez@rug.nl

ABSTRACT

Programming abstractions for *compensation handling* and *dynamic update* are crucial in specifying reliable interacting systems, such as Collective Adaptive Systems (CAS). Compensations and updates both specify how a system reacts in response to exceptional events. Prior work showed that different semantics for compensation handling can be encoded into a calculus of *adaptable processes* with *objective updates*, in which a process is reconfigured by its context. This paper goes further by considering *subjective updates*, in which, intuitively, a process reconfigures itself. A calculus of adaptable processes with subjective update its introduced, and its expressivity is assessed by encoding two semantics for compensation handling. The resulting encodings are more efficient than those using objective updates: they require less computational steps.

CCS Concepts

•Theory of computation → Semantics and reasoning; Process calculi; •Software and its engineering → Error handling and recovery;

Keywords

Concurrency, semantics of programming languages, process calculi, compensation handling, dynamic update, expressiveness.

1. INTRODUCTION

The staggering number of connected digital artifacts has given rise to 'systems of systems' that define the new socio-technical fabric of society. These Collective Adaptive Systems (CAS) are composed by massive number of units, each one having autonomous objectives and actions. The interaction patterns between these units hardly fit in known models and specification languages.

Our interest is in core programming models and linguistic constructs that define *self-adaptation* and *evolution* in settings such as CAS. Rather than defining new constructs, we seek to understand to what extent concepts from established paradigms (such as mobile communication and service-orientation) relate to each other. We see this as an essential prerequisite step towards the definition of sensible, widely applicable programming abstractions. Concretely, here we compare linguistic constructs for *compensation handling* and *dynamic update*. Following a process calculi approach, we formally relate them by studying their *relative expressiveness*.

Programming constructs that support failure handling at the heart of mechanisms that detect failures and bring the system back to a consistent state. In particular, *compensation primitives* install/activate alternative behaviors to compensate the fact that a (long-running) transaction has failed or has been aborted. Widely studied in service-oriented settings, forms of compensation handling also find an application in CAS (at least conceptually), especially as self-autonomous devices begin to be used in traditional transactional activities, such as distribution and delivery—consider, e.g., Amazon's Prime Air and DHL's Parcelcopter.

Several formal models with compensation primitives, endowed with different semantics, have been put forward. To clarify the relative expressiveness of many of these proposals, Lanese et al. [9] defined a core calculus of ***compensable processes***: it extends the π-calculus [12] with *transactions* $t[P\,,Q]$ (where P and Q represent default and compensation activities, respectively), *protected blocks* $\langle Q\rangle$, and *compensation updates* that reconfigure a compensation activity. Different proposals arise as instances of this calculus: compensations may admit *dynamic* or *static* recovery; nested transactions and protected blocks can be kept after failures via *preserving*, *discarding*, and *aborting* semantics. The language in [9] thus leads to six instances of calculi with compensation primitives.

Related to compensations but on a different vein, Bravetti et al. proposed a calculus of ***adaptable processes*** to specify the *dynamic update* in communicating systems [2]. Adaptable processes specify dynamic update as triggered by exceptional events, not necessarily catastrophic. For instance, the update of specific units of a robot swarm is usually hard to predict, and entails modifying the device's behavior; still, it is certainly not a failure. In adaptable processes, a *located process* $l[P]$ (where l is a location) can be reconfigured by an *update prefix* $l^o\{(X).Q\}.R$, an adaptation routine for l where variable X occurs zero or more times in Q. In $l[P]$, location l is *transparent*: $l[P]$ may behave as P. With these constructs, dynamic update is realized by the following reduction rule, in which contexts C_1 and C_2 denote arbitrary nested locations:

$$C_1\big[l[P]\big] \mid C_2\big[l^o\{(X).Q\}.R\big] \longrightarrow C_1\big[Q\{^P/_X\}\big] \mid C_2\big[R\big] \quad (1)$$

We call this ***objective update***: a located process is reconfigured by an update process in its context. This is a form of *process mobility* (higher-order communication [13]): process $l^o\{(X).Q\}$ moves from C_2 to C_1; the reconfigured behavior $Q\{^P/_X\}$ is left in C_1.

The purpose of this paper is to compare compensation handling (as formalized in [9]) and dynamic update (as formalized in [2]). Our prior work [7] encoded the six calculi in [9] into adaptable

SAC 2017, April 03 - 07, 2017, Marrakech, Morocco

ISBN 978-1-4503-4486-9/17/04…$15.00

DOI: http://dx.doi.org/10.1145/3019612.3019625

processes with objective update as in Rule (1). These encodings, however, are not as efficient as one may like: objective updates turn out to be inconvenient when "collecting" protected blocks scattered within nested transactions. Roughly, the problem is that objective update leaves processes in the "wrong location"; as a result, the encodings in [7] need additional adaptation steps to bring processes into appropriate locations and achieve the intended semantics. This difficulty reflects prominently in the number of reductions required to mimic a compensation step. That is, the encodings in [7] are inefficient because they are costly in terms of computational steps. Even if the encodings in [7] add only a few reductions per each compensation, the overall effect of such additional steps must be analyzed in the context of systems with many (nested) transactions, for which the cost of extra reduction steps may rapidly escalate.

More precisely, our prior work [7] does not address the following question: to what extent the *direction* of movement in (1) is responsible for the complexity and cost of the encodings? To clarify this issue, in this paper we encode compensation primitives into adaptable processes with ***subjective update***. This is a form of dynamic update not studied in [7], which implements the opposite direction of process movement implemented by objective update. Indeed, in subjective update a located process moves to a (remote) context containing an update process:

$$C_1\big[l[P] \mid R_1\big] \mid C_2\big[l^s\langle\!\langle (X).Q \rangle\!\rangle.R\big] \longrightarrow C_1\big[R_1\big] \mid C_2\big[Q\{^P/_X\} \mid R\big] \quad (2)$$

Above, $l^s\langle\!\langle (X).Q \rangle\!\rangle$ is a *subjective update prefix*. As objective update, subjective update relies on process mobility. However, the direction of movement is different: above, process P moves from C_1 to C_2; the updated behavior $Q\{^P/_X\}$ is left in C_2, not in C_1. Since the located process "updates itself", subjective updates offer a more autonomous adaptation semantics than objective updates.[1]

In this paper, we carefully revisit the encodings of compensation handling in [7] using subjective update (as in Rule (2)) rather than objective update (as in Rule (1)). The **main contribution** of the paper are therefore new, efficient encodings of the calculi in [9]. Using results of operational correspondence, we prove that these encodings with subjective update are correct and are more efficient than those with objective update proposed in [7].

The paper is organized as follows. § 2 illustrates primitives for compensation handling; § 3 formally presents the calculi of compensable processes and of adaptable processes. § 4 defines the notion of encoding. In § 5 we define and prove correct encodings of processes with (static) compensations into adaptable processes, considering discarding and aborting semantics. § 6 compares encodings into calculi with objective and subjective updates. § 7 collects concluding remarks and directions for future work.

2. COMPENSABLE PROCESSES

The process language with compensations we consider here is based on the calculus in [10] (a variant of the language in [9]). The languages in [9, 10] are extensions of the π-calculus [12] with primitives for *static* and *dynamic recovery*. As in [7], we consider variants of the languages in [9, 10] without name mobility and with static recovery. There are two salient constructs:

1. *Transactions* $t[P\,,Q]$, where t is a name and P, Q are processes;
2. *Protected blocks* $\langle Q \rangle$, for some process Q.

Basic Intuitions. A transaction $t[P\,,Q]$ consists of a *default activity* P with a *compensation activity* Q. Transactions can be nested: process P in $t[P\,,Q]$ may contain other transactions. Also, they can be aborted: intuitively, process $t[P\,,Q]$ behaves as P until an *error notification* (abortion signal) arrives along name t. Error notifications are output messages which can originate inside or outside the transaction. As an example, consider the following transitions:

$$t[P\,,Q] \mid \overline{t}.R \xrightarrow{\tau} Q \mid R \qquad\qquad t[\overline{t}.R \mid P_2\,,Q] \xrightarrow{\tau} Q \quad (3)$$

The left transition shows how t can be aborted by an external signal; the right transition illustrates internal abortion. Abortion discards the default behavior; the compensation activity is executed instead.

As their name suggests, *protected blocks* protect a process from abortion signals. Protected blocks are transparent: Q and $\langle Q \rangle$ have the same behavior, but $\langle Q \rangle$ is not affected by abortion signals. Protected blocks are meant to prevent abortions after a compensation:

$$t_2[P_2\,,Q_2] \mid \overline{t_2} \xrightarrow{\tau} \langle Q_2 \rangle$$

Consider now process $P = t_1\big[t_2[P_2\,,Q_2] \mid \overline{t_2}.R_1\,,Q_1\big]$, which includes a transaction t_2 which is *nested* inside t_1. Although in (3) the default behavior was erased following an abortion signal, the semantics of compensations may partially preserve such behavior. This preservation is realized by *extraction functions*, denoted $\mathtt{extr}(\cdot)$. For process P, the semantics in [9, 10] decree:

$$t_1\big[t_2[P_2\,,Q_2] \mid \overline{t_2}.R_1\,,Q_1\big] \xrightarrow{\tau} t_1\big[\langle Q_2 \rangle \mid \mathtt{extr}(P_2) \mid R_1\,,Q_1\big]$$

In case transaction t_2 is aborted, its compensation behavior Q_2 will be preserved. Moreover, part of the behavior of P_2 will be preserved as well: this is expressed by process $\mathtt{extr}(P_2)$, which consists of at least all protected blocks in P_2; it may also contain some other processes, related to transactions (see next). Here we consider *discarding* and *aborting* variants for $\mathtt{extr}(\cdot)$:

- $\mathtt{extr}_{\mathtt{D}}(P)$ keeps protected blocks at the top-level in P. Other processes are discarded.
- $\mathtt{extr}_{\mathtt{A}}(P)$ keeps all protected blocks in P, including protected blocks from all nested transactions in P and their respective compensation activities. Other processes are discarded.

Discarding and aborting semantics thus define different levels of protection for protected blocks. This way, e.g., given process $P = t\big[t_1[P_1\,,Q_1] \mid t_2[\langle P_2 \rangle\,,Q_2] \mid \langle P_3 \rangle\,,Q_5\big]$, where P_1 does not contain protected blocks, we would have:

$$\begin{array}{lll} \text{Discarding :} & \overline{t} \mid P & \xrightarrow{\tau}_{\mathtt{D}} \ \langle P_3 \rangle \mid \langle Q_5 \rangle \\ \text{Aborting :} & \overline{t} \mid P & \xrightarrow{\tau}_{\mathtt{A}} \ \langle P_3 \rangle \mid \langle Q_5 \rangle \mid \langle P_2 \rangle \mid \langle Q_1 \rangle \mid \langle Q_2 \rangle \end{array}$$

Thus, discarding semantics preserves only the compensation activity for t and the protected block $\langle P_3 \rangle$. Aborting semantics preserves all protected blocks and compensation activities in the default activity for t, including those in nested transactions.

Consider now a *Hotel booking scenario*, represented as compensable processes (in the following we omit trailing **0**s):

$$\begin{array}{rcl} R & \stackrel{def}{=} & Hotel \mid Client \\ Hotel & \stackrel{def}{=} & t[book.pay.\overline{invoice} \mid t_1[\langle DB \rangle\,,\mathbf{0}]\,,\overline{refund}] \\ Client & \stackrel{def}{=} & \overline{book}.\overline{pay}.(\overline{t}.refund + invoice) \end{array}$$

The hotel is modeled as a transaction t that allows clients to book a room and pay for it; t contains a database represented by process DB, kept within a protected block inside nested transaction t_1. If the client is satisfied with the reservation, then the hotel will send

[1] The terminology 'subjective' and 'objective' updates is inspired by the distinction between subjective and objective *mobility*, as in calculi such as Ambients [3] and Seal [4]. As explained in [4], Ambients use subjective mobility (an agent moves itself), while Seal uses objective mobility (an agent is moved by its context).

him an invoice. Otherwise, the client may abort the transaction; in that case, hotel offers the client a refund.

Suppose that the client decides to abort his reservation; the transitions for R under discarding and aborting semantics are:

$$R \xrightarrow{\tau}{}^{*}_{\mathtt{D}} \langle\overline{refund}\rangle \mid refund \xrightarrow{\tau}_{\mathtt{D}} \langle\mathbf{0}\rangle \equiv \mathbf{0}$$
$$R \xrightarrow{\tau}{}^{*}_{\mathtt{A}} \langle DB\rangle \mid \langle\overline{refund}\rangle \mid refund \xrightarrow{\tau}_{\mathtt{A}} \langle DB\rangle.$$

Here again it is easy to see how, in case of compensation, an aborting semantics preserves protected blocks in nested transactions.

3. THE CALCULI

We introduce adaptable processes (§ 3.1) and compensable processes (§ 3.2). To focus on their essentials, both calculi are defined as extensions of CCS [11] (no name passing). In both cases, we assume a countable set of names *N*, ranged over by $a, b, l, t, \ldots$. We use names $l, l', \ldots$ to denote locations (in adaptable processes) and names $t, t', \ldots$ to denote transactions (in compensable processes).

3.1 Adaptable Processes

The syntax of the calculus of *adaptable processes* is defined by *prefixes* $\pi, \pi', \ldots$, defined as $\pi ::= a \mid \overline{a} \mid l^s\langle\!\langle(X).Q\rangle\!\rangle$ and *processes* $P, Q, \ldots$ defined as

$$P ::= l[P] \mid \mathbf{0} \mid \pi.P \mid !P \mid P \mid Q \mid (\nu a)P \mid X$$

We consider input and output prefixes (noted a and $\overline{a}$) and the *subjective update prefix* $l^s\langle\!\langle(X).Q\rangle\!\rangle$, where Q may contain zero or more occurrences of *process variable* X. For simplicity, we will write $l\langle\!\langle(X).Q\rangle\!\rangle$ instead of $l^s\langle\!\langle(X).Q\rangle\!\rangle$. The syntax of processes includes *located processes* (noted $l[P]$ and already motivated) as well as usual CCS constructs for inaction, prefix (sequentiality), replication, parallel composition, and restriction. We omit $\mathbf{0}$ whenever possible; we write, e.g., $l\langle\!\langle(X).P\rangle\!\rangle$ instead of $l\langle\!\langle(X).P\rangle\!\rangle.\mathbf{0}$.

Name a is bound in $(\nu a)P$ and process variable X is bound in $l\langle\!\langle(X).Q\rangle\!\rangle$; given a process P, its sets of free and bound names/variables (denoted $\mathtt{fn}(P)$, $\mathtt{bn}(P)$, $\mathtt{fv}(P)$, and $\mathtt{bv}(P)$) are as expected. We rely on usual notions of α-conversion (noted $\equiv_\alpha$) and process substitution: $P\{^Q/_X\}$ denotes the process obtained by (capture avoiding) substitution of Q for X in P.

The semantics of adaptable processes is given by a reduction semantics, which relies on *structural congruence*, denoted $\equiv$, and *contexts*, denoted C, D, E. We define $\equiv$ as the smallest congruence on processes that includes $\equiv_\alpha$ and satisfies the following axioms:

$$P \mid Q \equiv Q \mid P \quad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \quad P \mid \mathbf{0} \equiv P$$
$$(\nu a)\mathbf{0} \equiv \mathbf{0} \qquad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \qquad !P \equiv P \mid !P$$
$$(\nu a)P \mid Q \equiv (\nu a)(P \mid Q) \text{ if } a \notin \mathtt{fn}(Q)$$
$$(\nu a)l[P] \equiv l[(\nu a)P]$$

Contexts are processes with a *hole* $[\bullet]$. Their syntax is defined as:

$$C ::= [\bullet] \mid C \mid P \mid l[C].$$

We write $C[P]$ to denote the process obtained by replacing the hole $[\bullet]$ in context C with P.

Reduction $\rightarrow$ is the smallest relation on processes induced by the rules in Figure 1, which we now briefly discuss. Rule (R-I/O) formalizes synchronization between processes $\overline{a}.P$ and $a.Q$, enclosed in contexts C and D, respectively. Rule (R-SubUpd) formalizes the subjective update of a location l, as motivated in the Introduction. Rules (R-Par), (R-Loc), (R-Res), and (R-Str) are standard and/or self-explanatory. We write $\rightarrow^*$ to denote the reflexive, transitive closure of $\rightarrow$.

(R-I/O)
$$E\Big[C\big[\overline{a}.P\big] \mid D\big[a.Q\big]\Big] \rightarrow E\Big[C\big[P\big] \mid D\big[Q\big]\Big]$$

(R-SubUpd)
$$E\Big[C\big[l[P]\big] \mid D\big[l^s\langle\!\langle(X).Q\rangle\!\rangle.R\big]\Big] \rightarrow E\Big[C\big[\mathbf{0}\big] \mid D\big[Q\{^P/_X\} \mid R\big]\Big]$$

(R-Par) $\dfrac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}$ (R-Str) $\dfrac{P \equiv P' \;\; P' \rightarrow Q' \;\; Q' \equiv Q}{P \rightarrow Q}$

(R-Res) $\dfrac{P \rightarrow P'}{(\nu a)P \rightarrow (\nu a)P'}$ (R-Loc) $\dfrac{P \rightarrow P'}{l[P] \rightarrow l[P']}$

Figure 1: Reduction semantics for adaptable processes.

3.2 Compensable Processes

Building upon input and output prefixes $\pi ::= a \mid \overline{a}$, the calculus of *compensable processes* is defined by the following syntax:

$$P, Q ::= \mathbf{0} \mid \pi.P \mid !P \mid (\nu a)P \mid P \mid Q \mid t[P\,,Q] \mid \langle Q\rangle$$

Processes for inaction ($\mathbf{0}$), sequentiality ($\pi.P$), replication ($!P$), restriction ($(\nu a)P$), and parallel composition ($P \mid Q$) are standard. We omit $\mathbf{0}$ whenever possible. Protected blocks $\langle Q\rangle$ and transactions $t[P\,,Q]$ have been already motivated. Error notifications are output messages; they can be internal or external to the transaction. Name a is bound in $(\nu a)P$; α-conversion (noted $\equiv_\alpha$) is as expected. We assume that protected blocks and transactions do not appear behind prefixes; this is key to ensure encoding correctness.

Following [9, 10], the semantics of compensable processes is given in terms of a Labeled Transition System (LTS). Ranged over by α, α', the set of labels includes a, $\overline{a}$ and τ. As in CCS, a denotes an input action, $\overline{a}$ denotes an output action, and τ denotes synchronization (internal action). Formally, we have two different LTSs, corresponding to processes under discarding and aborting semantics. For each $\kappa \in \{\mathtt{D}, \mathtt{A}\}$, we will have an extraction function $\mathtt{extr}_\kappa(\cdot)$ and a transition relation $\xrightarrow{\alpha}_\kappa$. The different extraction functions are defined in Fig. 2; the rules of the LTSs are given in Fig. 3. As a convention, whenever a notion coincides for the two semantics, we avoid decorations $\mathtt{D}$ and $\mathtt{A}$.

We comment on the rules in Fig. 3:

- Axioms (L-Out) and (L-In) execute output and input prefixes, respectively.
- Rule (L-Par) allows one parallel component to progress independently.
- Rule (L-Res) is the standard rule for restriction. A transition of process P determines a transition of process $(\nu a)P$, where label α provided that the restricted name a does not occur inside α.
- Rule (L-Comm) defines communication on a.
- Rule (L-Scope-Out) allows the default activity P of a transaction to progress.
- Rule (L-Recover-Out) allows an external process to abort a transaction via an output action $\overline{t}$. The resulting process contains two parts: the first is obtained from the default activity P of the transaction using the extraction function; the second part corresponds to compensation Q, executed in a protected block.
- Rule (L-Recover-In) handles abortion when the error notification comes from the default activity P of the transaction.

$$\begin{aligned}
\mathsf{extr}(\pi.P) = \mathsf{extr}(!P) = \mathsf{extr}(\mathbf{0}) &= \mathbf{0}\\
\mathsf{extr}(\langle P\rangle) &= \langle P\rangle\\
\mathsf{extr}(P \mid Q) &= \mathsf{extr}(P) \mid \mathsf{extr}(Q)\\
\mathsf{extr}_{\mathtt{D}}(t[P\,,Q]) &= \mathbf{0}\\
\mathsf{extr}_{\mathtt{A}}(t[P\,,Q]) &= \mathsf{extr}_{\mathtt{A}}(P) \mid \langle Q\rangle\\
\mathsf{extr}((\nu a)P) &= (\nu a)\mathsf{extr}(P)
\end{aligned}$$

Figure 2: Extraction functions for compensable processes.

$$\text{(L-Out)}\ \ \overline{a}.P \xrightarrow{\overline{a}} P \qquad \text{(L-In)}\ \ a.P \xrightarrow{a} P$$

$$\text{(L-Par)}\ \ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \text{(L-Res)}\ \ \frac{P \xrightarrow{\alpha} P' \ \ \alpha \neq a, \overline{a}}{(\nu a)P \xrightarrow{\alpha} (\nu a)P'}$$

$$\text{(L-Comm)}\ \ \frac{P \xrightarrow{a} P' \ \ Q \xrightarrow{\overline{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{(L-Block)}\ \ \frac{P \xrightarrow{\alpha} P'}{\langle P\rangle \xrightarrow{\alpha} \langle P'\rangle}$$

$$\text{(L-Recover-Out)}\ \ t[P\,,Q] \xrightarrow{t} \mathsf{extr}_{\kappa}(P) \mid \langle Q\rangle \qquad \text{(L-Scope-Out)}\ \ \frac{P \xrightarrow{\alpha} P'}{t[P\,,Q] \xrightarrow{\alpha} t[P'\,,Q]}$$

$$\text{(L-Recover-In)}\ \ \frac{P \xrightarrow{\overline{t}} P'}{t[P\,,Q] \xrightarrow{\tau} \mathsf{extr}_{\kappa}(P') \mid \langle Q\rangle} \qquad \text{(L-Rep)}\ \ \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P}$$

Figure 3: LTS for compensable processes. We omit symmetric variants of (L-Par) and (L-Comm).

- Rule (L-Rep) deals with replication, while Rule (L-Block) essentially specifies that protected blocks are transparent units.
- The semantics of protected blocks is defined via the extraction functions $\mathsf{extr}(\cdot)$ (see Fig. 2).

It is convenient to define structural congruence ($\equiv$) and contexts also for compensable processes. We define $\equiv$ as the smallest congruence on processes that includes $\equiv_\alpha$ and satisfies the axioms:

$$\begin{array}{l}
P \mid Q \equiv Q \mid P \quad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \quad P \mid \mathbf{0} \equiv P\\
(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \quad (\nu a)\mathbf{0} \equiv \mathbf{0} \quad (\nu a)\overline{a} \equiv \mathbf{0}\\
\langle\langle P\rangle\rangle \equiv \langle P\rangle \quad \langle (\nu a)P\rangle \equiv (\nu a)\langle P\rangle \quad \langle\mathbf{0}\rangle \equiv \mathbf{0}\\
t[(\nu a)P\,,Q] \equiv (\nu a)t[P\,,Q] \text{ if } t \neq a,\ a \notin \mathtt{fn}(Q)\\
(\nu a)P \mid Q \equiv (\nu a)(P \mid Q) \text{ if } a \notin \mathtt{fn}(Q)
\end{array}$$

An n-adic context $C[\bullet_1, \ldots, \bullet_n]$ is obtained from a process by replacing n occurrences of $\mathbf{0}$, that are neither compensations nor in continuation of prefixes, with indexed holes $[\bullet_1], \ldots, [\bullet_n]$. This way, for instance, the syntax of (monadic) contexts is defined as:

$$C ::= [\bullet] \ \Big|\ \langle C\rangle \ \Big|\ t[C\,,Q] \ \Big|\ P \mid C \ \Big|\ C \mid P \ \Big|\ (\nu a)C.$$

We write $C[P]$ to denote the process obtained by replacing the hole $[\bullet]$ in context C with P. The following proposition is key to our operational correspondence statements.

Proposition 3.1. *Let P be a compensable process. If $P \xrightarrow{\tau} P'$ then one of the following holds:*

(a) $P \equiv E[C[\overline{a}.P_1] \mid D[a.P_2]]$ and $P' \equiv E[C[P_1] \mid D[P_2]]$,

(b) $P \equiv E[C[t[P_1\,,Q]] \mid D[\overline{t}.P_2]]$ and $P' \equiv E[C[\mathsf{extr}_{\kappa}(P_1) \mid \langle Q\rangle] \mid D[P_2]]$, or

(c) $P \equiv C[t[D[\overline{t}.P_1]\,,Q]]$ and $P' \equiv C[\mathsf{extr}_{\kappa}(D[P_1]) \mid \langle Q\rangle]$.

for some contexts C, D, E, processes P_1, P_2, Q, and names a, t.

4. THE NOTION OF ENCODING

We relate compensable and adaptable processes through *encodings*. A (valid) encoding is a translation of processes of a *source language* into the processes of a *target language* that satisfies certain *correctness criteria*, which attest to the encoding's quality. The existence of a valid encoding shows that the target language is at least as expressive as the source language. Conversely, proving the non existence of such an encoding shows that the source language can express some behavior not expressible in the target language. By combining these *positive* and *negative encodability results*, differences in expressivity between languages can be established.

To define valid encodings, we rely on the abstract formulation of [8], focusing on compositionality and operational correspondence criteria. Following [8], a *calculus* is a triple $(\mathcal{P}, \rightarrow, \approx)$, where $\mathcal{P}$ is a set of processes, $\rightarrow$ is its operational semantics, and $\approx$ is a behavioral equivalence. A *valid encoding* of the source calculus $(\mathcal{P}_1, \rightarrow_1, \approx_1)$ into the target calculus $(\mathcal{P}_2, \rightarrow_2, \approx_2)$ is then a mapping $[\![\cdot]\!] : \mathcal{P}_1 \rightarrow \mathcal{P}_2$ that satisfies some specific criteria.

In the following, given an operational semantics $\rightarrow$ and $k \geq 1$, we will write $\rightarrow^k$ to denote k reduction steps. Also, we write $\rightarrow^*$ to denote the reflexive, transitive closure of $\rightarrow$.

Definition 4.1. *A translation (mapping) $[\![\cdot]\!] : \mathcal{P}_1 \rightarrow \mathcal{P}_2$ is a* valid encoding *if it satisfies the following criteria:*

1. Compositionality*: $[\![\cdot]\!]$ is* compositional *if for every n-ary operator $\mathtt{op}$ on $\mathcal{P}_1$ there is an n-adic context $C_{\mathtt{op}}$ in $\mathcal{P}_2$ such that $[\![\mathtt{op}(P_1, \ldots, P_n)]\!] = C_{\mathtt{op}}\,([\![P_1]\!], \ldots, [\![P_n]\!])$.*
2. Operational correspondence*, divided into two requirements:*
   a) *Completeness: If $P \rightarrow_1 Q$ then there are P' and k such that $[\![P]\!] \rightarrow_2^k P' = [\![Q]\!]$.*
   b) *Soundness: If $[\![P]\!] \rightarrow_2^* R$ then there is P' such that $P \rightarrow_1^* P'$ and $R \rightarrow_2^* [\![P']\!]$.*

In this paper, the source languages will be compensable processes with discarding and aborting semantics; the target language will be adaptable processes with subjective update. The operational semantics for compensable processes will be τ-labeled transitions; for adaptable processes we consider their reduction semantics. Also, we write $\xrightarrow{\tau}{}^*$ to denote the reflexive, transitive closure of $\xrightarrow{\tau}$.

5. COMPENSABLE PROCESSES INTO ADAPTABLE PROCESSES

We define translations of compensable processes into adapta ble processes, and show that they are valid encodings. We focus on compensable processes with discarding and aborting semantics, using static recovery. Given the encodings presented here, extensions to preserving semantics and dynamic recovery are simple.

Conventions. We shall write $\mathcal{A}$ to denote adaptable processes with subjective update, as in § 3.1. Also, we shall write $\mathcal{C}_{\mathtt{D}}$ and $\mathcal{C}_{\mathtt{A}}$ to denote compensable processes with discarding and aborting semantics, respectively, as defined in § 3.2. For convenience, we shall adopt the following abbreviations for adaptable processes; below, C_1, C_2 are contexts, P, Q, R are processes and t is location.

- $t^s\langle\!\langle\dagger\rangle\!\rangle$ stands for the update prefix $t^s\langle\!\langle(Y).\mathbf{0}\rangle\!\rangle$ which "kills" both location t and the process located at t. For instance:
$$C_1\big[t[P_1] \mid Q\big] \mid C_2\big[t^s\langle\!\langle\dagger\rangle\!\rangle.R\big] \rightarrow C_1\big[\mathbf{0} \mid Q\big] \mid C_1\big[R\big]$$
- We write $t^s\langle\!\langle(Y_1, Y_2, \ldots, Y_n).R\rangle\!\rangle$ to abbreviate the nested updates $t^s\langle\!\langle(Y_1).t^s\langle\!\langle(Y_2).\cdots.t^s\langle\!\langle(Y_n).R\rangle\!\rangle\cdots\rangle\!\rangle\rangle\!\rangle$. For instance,

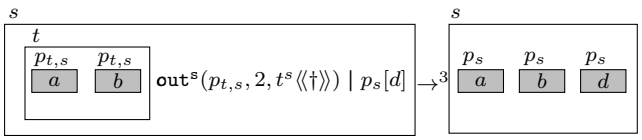

Figure 4: Example of $\mathtt{out}^{\mathtt{s}}(p_{\rho_0}, n, Q)$.

let $S = C_1\big[t[P] \mid t[Q]\big] \mid C_2\big[t^s\langle\langle(Y_1).t^s\langle\langle(Y_2).R\rangle\rangle\rangle\big]$. Then

$$\begin{aligned} S &\rightarrow C_1\big[\mathbf{0} \mid t_2[Q]\big] \mid C_2\big[t^s\langle\langle(Y_2).R\{^P/_{Y_1}\}\rangle\rangle\big] \\ &\rightarrow C_1\big[\mathbf{0} \mid \mathbf{0}\big] \mid C_2\big[R\{^P/_{Y_1}\}\{^Q/_{Y_2}\}\big] \end{aligned}$$

- We write $\prod_{i=1}^{n} l[X_i]$ to denote the process $l[X_1] \mid \ldots \mid l[X_n]$.

Well-formed Processes. To give a sharp formulation of operational correspondence (in particular, soundness) for our encodings, we shall focus on a class of *well-formed* compensable processes, which excludes processes with dangerous combinations of nested transactions and concurrent abortion signals. A concise example of a process that is not well-formed is the following:

$$P = t_1\big[P_1 \mid t_2[P_2\,,Q_2]\,,Q_1\big] \mid \overline{t_1} \mid \overline{t_2}.$$

Processes such as P feature a complex form of non determinism that it is hard to capture properly in the (lower level) representation that we shall give in terms of adaptable processes. Indeed, P represents an *interference* between the abortion of t_1 and t_2; it is hard to imagine patterns where this kind of interfering concurrency may come in handy. In contrast, the following variants of P are well-formed and are faithfully translated by our encodings:

$$t_1\big[P_1 \mid t_2[P_2\,,Q_2]\,,Q_1\big] \mid \overline{t_1}.\overline{t_2}$$
$$t_1[P_1\,,Q_1] \mid t_2[P_2\,,Q_2] \mid \overline{t_1} \mid \overline{t_2}.$$

5.1 Discarding Semantics

The distinguishing constructs in the calculus of compensable processes are transactions and protected blocks; they represent the most interesting process terms to be handled by our encodings.

To encode protected blocks we use a reserved name p and associate it with a *path* ρ: a sequence of (nested) location names representing the place of the protected block in the tree representation of a process. We write ϵ to denote the empty path. In all the non empty paths ρ we will omit ϵ at the end. In this way, the encoding of a protected block found at path ρ, is defined as

$$[\![\langle P\rangle]\!]^{\mathtt{D}}_{\rho} = p_\rho\big[[\![P]\!]^{\mathtt{D}}_{\epsilon}\big].$$

To encode the extraction function, essential in the semantics of compensable processes (cf. Figure 2), we use an auxiliary process, denoted $\mathtt{out}^{\mathtt{s}}(p_{t,\rho}, n, Q)$, that moves n processes from locations $p_{t,\rho}$ to locations p_ρ, and puts Q in parallel. Given $n > 0$, this process can be efficiently defined using subjective update prefixes:

$$\mathtt{out}^{\mathtt{s}}(p_{t,\rho}, n, Q) = p^s_{t,\rho}\langle\langle(X_1, \ldots, X_n).\big(\prod_{i=1}^{n} p_\rho[X_i] \mid Q\big)\rangle\rangle \quad (4)$$

We define $\mathtt{out}(p_{t,\rho}, n, Q) = Q$ if $n = 0$. This way, e.g., we have

$$\begin{aligned} s\Big[t\big[p_{t,s}[a] \mid p_{t,s}[b]\big] \mid \mathtt{out}^{\mathtt{s}}(p_{t,s}, 2, t^s\langle\langle\dagger\rangle\rangle) \mid p_s[d]\Big] \\ \rightarrow^3 s\big[p_s[a] \mid p_s[b] \mid p_s[d]\big] \end{aligned}$$

as illustrated in Figure 4 (with omitted trailing occurrences of $\mathbf{0}$).

5.1.1 The Translation

Before presenting the translation $[\![\cdot]\!]^{\mathtt{D}}_{\rho} : \mathcal{C}_{\mathtt{D}} \rightarrow \mathcal{A}$, we present two auxiliary definitions. First, we use the following function for determining the number of locations in a process:

$$\begin{aligned} \mathtt{nl}(l_1, l_2[P]) &= \mathtt{nl}(l_1, P) + 1 \text{ if } l_1 = l_2 \\ \mathtt{nl}(l_1, l_2[P]) &= \mathtt{nl}(l_1, P) \text{ if } l_1 \neq l_2 \\ \mathtt{nl}(l, (\nu a)\,P) &= \mathtt{nl}(l, P) \\ \mathtt{nl}(l, P \mid Q) &= \mathtt{nl}(l, P) + \mathtt{nl}(l, Q) \\ \mathtt{nl}(l, \mathbf{0}) &= \mathtt{nl}(l, !P) = \mathtt{nl}(l, \pi.P) = 0 \end{aligned}$$

Second, let t and l be names; we use process $\mathtt{extr}_{\mathtt{D}}(t, l)$, defined as

$$t^s\langle\langle(Y).t[Y] \mid \mathtt{out}^{\mathtt{s}}(l, \mathtt{nl}(l, Y), t^s\langle\langle\dagger\rangle\rangle.\overline{h_t})\rangle\rangle. \quad (5)$$

We then have the following definition:

DEFINITION 5.1. *Let ρ be a path. Also, let h_t and p_ρ be fresh names. We define the translation $[\![\cdot]\!]^{\mathtt{D}}_{\rho} : \mathcal{C}_{\mathtt{D}} \rightarrow \mathcal{A}$ as*

$$\begin{aligned} [\![t[P\,,Q]]\!]^{\mathtt{D}}_{\rho} &= t\Big[[\![P]\!]^{\mathtt{D}}_{t,\rho}\Big] \mid t.\,\big(\mathtt{extr}_{\mathtt{D}}(t, p_{t,\rho}) \mid p_\rho[[\![Q]\!]^{\mathtt{D}}_{\epsilon}]\big) \\ [\![\langle P\rangle]\!]^{\mathtt{D}}_{\rho} &= p_\rho\big[[\![P]\!]^{\mathtt{D}}_{\epsilon}\big] \\ [\![t.P]\!]^{\mathtt{D}}_{\rho} &= t.\overline{h_t}.[\![P]\!]^{\mathtt{D}}_{\rho} \\ [\![\,\overline{t}.P\,]\!]^{\mathtt{D}}_{\rho} &= \overline{t}.h_t.[\![P]\!]^{\mathtt{D}}_{\rho} \end{aligned}$$

and as a homomorphism for other operators.

A key aspect in our encodings concerns the extraction function. For compensable processes, the extraction function is an external semantic device used to formalize the protection of transactions/protected blocks; in fact, it is not formally modeled by process terms. In contrast, our encodings explicitly specify the essence of extraction functions by means of (subjective) update prefixes.

Path t, ρ states that t is nested in the transactions listed in ρ. In case of an abortion signal $\overline{t}$, process $\mathtt{extr}(t, p_{t,\rho})$ will extract all processes located at $p_{t,\rho}$ (which are encodings of protected blocks). Since the structure of a transaction and the number of its top-level processes dynamically changes, whenever we need to extract processes located at $p_{t,\rho}$, we will first substitute Y, in process $\mathtt{out}$, with the content of the location t and count the current number of locations $p_{t,\rho}$. For example, let $\tilde{i}$ stands for $i.\overline{h_i}$, with $i \in \{a, b, d\}$, we have:

$$\begin{aligned} & s\big[t[p_{t,s}[\tilde{a}] \mid p_{t,s}[\tilde{b}]] \mid \mathtt{extr}_{\mathtt{D}}(t, p_{t,s})\big] \\ & \rightarrow s\big[t[p_{t,s}[\tilde{a}] \mid p_{t,s}[\tilde{b}]] \\ & \quad \mid \mathtt{out}^{\mathtt{s}}(p_{t,s}, \mathtt{nl}(p_{t,s}, p_{t,s}[\tilde{a}] \mid p_{t,s}[\tilde{b}]), t^s\langle\langle\dagger\rangle\rangle.\overline{h_t}) \mid p_s[\tilde{d}]\big] \\ & \rightarrow s\big[t[p_{t,s}[\tilde{a}] \mid p_{t,s}[\tilde{b}]] \mid \mathtt{out}^{\mathtt{s}}(p_{t,s}, 2, t^s\langle\langle\dagger\rangle\rangle.\overline{h_t}) \mid p_s[\tilde{d}]\big]. \end{aligned}$$

Consider the encoding of $t[P\,,Q]$: if the abortion signal $\overline{t}$ is activated, after synchronizations on t, it will extract all processes at locations $p_{t,\rho}$ and move them to their parent location p_ρ, together with the encoding of the compensation activity Q. We need to leave only "garbage" at location t and to erase it together with the location. To this end, we use prefix $t^s\langle\langle\dagger\rangle\rangle$. Name h_t is introduced to control execution of abortion signals; it is particularly useful to represent for errors that occur sequentially.

5.1.2 Translation Correctness

The translation $[\![\cdot]\!]^{\mathtt{D}}_{\rho}$ is a valid encoding. The first property, compositionality, follows directly from its definition:

THEOREM 5.2 (COMPOSITIONALITY). *The translation $[\![\cdot]\!]^{\mathtt{D}}_{\rho}$ is compositional, in the sense of Def. 4.1(1).*

$$\begin{aligned}
\mathtt{pb}(\langle P\rangle) &= 1\\
\mathtt{pb}_{\mathtt{A}}(t[P\,,Q]) &= 1+\mathtt{pb}_{\mathtt{A}}(P)\\
\mathtt{pb}(P\mid Q) &= \mathtt{pb}(P)+\mathtt{pb}(Q)\\
\mathtt{pb}((\nu a)P) &= \mathtt{pb}(P)\\
\mathtt{pb}(!P) &= \mathtt{pb}_{\mathtt{D}}(t[P\,,Q]) = 0\\
\mathtt{pb}(\mathbf{0}) &= \mathtt{pb}(\pi.P) = 0
\end{aligned}$$

Figure 5: Number of protected blocks.

We are interested in giving a precise account of the number of computation steps used by our encodings. This is to support our claim that subjective updates are more efficient than objective updates. To this end, we introduce some auxiliary notions. Given a process P, we will write $\mathtt{pb}_{\mathtt{D}}(P)$ to denote the number of protected blocks in P; see Figure 5. There is a correspondence between (i) the number of protected blocks in the default activity of the source transaction and (ii) to the number of locations in the encoding of such a transaction. Also, in order to distinguish usual input/output synchronizations from compensation steps, we write $P \xrightarrow{\tau_a} P'$ if $P \xrightarrow{\tau} P'$ corresponds to a standard synchronization, for some name a (cf. Proposition 3.1(a)).

We then have the following result:

THEOREM 5.3 (OPERATIONAL CORRESPONDENCE). *Let P be a well-formed process in $\mathcal{C}_{\mathtt{D}}$. We have:*

1. *If $P \xrightarrow{\tau_a}_{\mathtt{D}} P'$ then $[\![P]\!]^{\mathtt{D}}_{\epsilon} \to^2 [\![P']\!]^{\mathtt{D}}_{\epsilon}$, for some a.*
2. *If $P \xrightarrow{\tau}_{\mathtt{D}} P'$ then $[\![P]\!]^{\mathtt{D}}_{\epsilon} \to^k [\![P']\!]^{\mathtt{D}}_{\epsilon}$ where either*
   a) $P \equiv E[C[t[P_1\,,Q]] \mid D[\overline{t}.P_2]]$ and $k = 4+\mathtt{pb}_{\mathtt{D}}(P_1)$ or
   b) $P \equiv C[t[D[\overline{t}.P_1]\,,Q]]$ and $k = 4+\mathtt{pb}_{\mathtt{D}}(D[P_1])$,
   for some contexts C, D, E, processes P_1, Q, P_2, and name t.
3. *If $[\![P]\!]^{\mathtt{D}}_{\epsilon} \to^* R$ then there is P' such that $P \xrightarrow{\tau}{}^*_{\mathtt{D}} P'$ and $R \to^* [\![P']\!]^{\mathtt{D}}_{\epsilon}$.*

Cases (1) and (2) concern completeness, while Case (3) describes soundness. Case (1) concerns standard synchronizations which are translated by $[\![\cdot]\!]^{\mathtt{D}}_{\rho}$ with an additional synchronization (on name h_t). Case (2) concerns synchronizations due to compensation signals; here the analysis distinguishes two cases, depending on whether the abortion signal is external or internal to the transaction. In both cases, the number of reduction steps required to mimic the source transition depends on the number of protected blocks of the transaction being aborted. However, internal abortion signals can be more efficiently mimicked than external abortion signals.

EXAMPLE 5.4. *Let $P = s\big[t[\langle a\rangle \mid \langle b\rangle \mid c\,,d]\,,\mathbf{0}\big] \mid \overline{t}.\overline{s}$ be a compensable process. By the LTS of Fig. 3, we have*

$$P \xrightarrow{\tau}_{\mathtt{D}} s[\langle a\rangle \mid \langle b\rangle \mid \langle d\rangle\,,\mathbf{0}] \mid \overline{s} \xrightarrow{\tau}_{\mathtt{D}} \langle a\rangle \mid \langle b\rangle \mid \langle d\rangle.$$

Expanding Def. 5.1 and using $\tilde{i}$ to stand for $i.\overline{h_i}$ ($i \in \{a,b,c,d\}$), we have:

$$\begin{aligned}
&s\Big[t\Big[p_{t,s}\big[\tilde{a}\big] \mid p_{t,s}\big[\tilde{b}\big] \mid \tilde{c}\Big] \mid t.\big(\mathtt{extr}_{\mathtt{D}}(t,p_{t,s}) \mid p_s\big[\tilde{d}\big]\big)\Big] \mid \\
&\qquad\qquad s.\mathtt{extr}_{\mathtt{D}}(s,p_s) \mid \overline{t}.h_t.\overline{s}.h_s\\
&\to^6 s\Big[p_s[\tilde{a}] \mid p_s\big[\tilde{b}\big] \mid p_s\big[\tilde{d}\big]\Big] \mid s.\mathtt{extr}_{\mathtt{D}}(s,p_s) \mid \overline{s}.h_s\\
&\to^7 p_\epsilon[\tilde{a}] \mid p_\epsilon\big[\tilde{b}\big] \mid p_\epsilon\big[\tilde{d}\big] = [\![\langle a\rangle]\!]^{\mathtt{D}}_{\epsilon} \mid [\![\langle b\rangle]\!]^{\mathtt{D}}_{\epsilon} \mid [\![\langle d\rangle]\!]^{\mathtt{D}}_{\epsilon}.
\end{aligned}$$

EXAMPLE 5.5. *We apply the encoding on the example of § 2, assuming that the client cancels after booking and paying:*

$$\begin{aligned}
[\![R]\!]^{\mathtt{D}}_{\epsilon} = \;& t\Big[book.\overline{h_b}.pay.\overline{h_p}.invoice.\overline{h_i} \mid t_1\big[p_{t_1,t}[[\![DB]\!]^{\mathtt{D}}_{\epsilon}]\big]\\
&\mid t_1.\big(\mathtt{extr}_{\mathtt{D}}(t_1,p_{t_1,t})\big)\Big] \mid t.(\mathtt{extr}_{\mathtt{D}}(t,p_t)\\
&\mid p_t[\overline{refund}.h_r])\\
&\mid \overline{book}.h_b.\overline{pay}.h_p.\overline{t}.h_t.refund.\overline{h_r}\\
\to^5\; & t\Big[invoice.\overline{h_i} \mid t_1[p_{t_1,t}[[\![DB]\!]^{\mathtt{D}}_{\epsilon}]]\\
&\mid t_1.\big(\mathtt{extr}_{\mathtt{D}}(t_1,p_{t_1,t})\big)\Big]\\
&\mid t^s\langle\!\langle (Y).t[Y] \mid \mathtt{out}^{\mathtt{s}}(p_t,\mathtt{nl}(p_t,Y),t^s\langle\!\langle\dagger\rangle\!\rangle.\overline{h_t})\rangle\!\rangle\\
&\mid p_t[\overline{refund}.h_r] \mid h_t.refund.\overline{h_r}\\
\to\; & t\Big[invoice.\overline{h_i} \mid t_1[p_{t_1,t}[[\![DB]\!]^{\mathtt{D}}_{\epsilon}]]\\
&\mid t_1.\big(\mathtt{extr}_{\mathtt{D}}(t_1,p_{t_1,t})\big)\Big] \mid \mathtt{out}^{\mathtt{s}}(p_t,0,t^s\langle\!\langle\dagger\rangle\!\rangle.\overline{h_t})\\
&\mid p_t[\overline{refund}.h_r] \mid h_t.refund.\overline{h_r} \to^4 p_t[\mathbf{0}] = [\![\mathbf{0}]\!]^{\mathtt{D}}_{\epsilon}.
\end{aligned}$$

Therefore we get $[\![R]\!]^{\mathtt{D}}_{\epsilon} \to^{10} p_t[[\![\mathbf{0}]\!]^{\mathtt{D}}_{\epsilon}]$.

5.2 Aborting Semantics

We now discuss the encoding of $\mathcal{C}_{\mathtt{A}}$ into $\mathcal{A}$. Aborting semantics keeps not only top-level protected blocks of a transaction, but also protected blocks from nested transactions (cf. Fig. 2). For this reason, given a located process, we define its *activation process*: this a process that captures the hierarchical structure of the nested locations of the process, which arise as a result of encoding transaction with its corresponding nested transactions. Names of nested locations that make up the activation process originate exclusively from its corresponding transaction name and the names of its nested transactions (i.e. the names of the locations that are of form p_ρ are not included in the formation of the activation process)

DEFINITION 5.6 (ACTIVATION PROCESS). *Let $l[P]$ be a located process. We denote by $St(P)$ the labeled tree (with root l) in which nodes are labeled with names of located processes and sub-trees capture nested locations. The* activation process *for P, denoted $\mathcal{T}_l(P)$, is the process obtained by a post-order search in $St(P)$ in which the visit to a node labeled c_i adds prefixes $\overline{l_{c_i}}.k_{c_i}$.*

This way, e.g., given $l[P]$ where

$$P = a[c[p_\rho[P_1]] \mid P_2] \mid b[P_3 \mid d[P_4] \mid e[P_5]]$$

and $P_1, \ldots, P_5$ do not contain located processes, we will have the activation process $\mathcal{T}_l(P) = \overline{l_c}.k_c.\overline{l_a}.k_a.\overline{l_d}.k_d.\overline{l_e}.k_e.\overline{l_b}.k_b.\overline{l_l}.k_l$.

Since an abortion signal extracts all nested protected blocks, and erases nested locations, our encoding will do the same with corresponding located processes and nested locations.

5.2.1 The Translation

Before introducing the translation $[\![\cdot]\!]^{\mathtt{A}}_{\rho} : \mathcal{C}_{\mathtt{A}} \to \mathcal{A}$, we give some addtional explanations. In this encoding we use the function for determining the number of locations in a process ($\mathtt{nl}$) as in § 5.1.1. For this translation, process $\mathtt{extr}_{\mathtt{A}}(t,l)$ is defined as:

$$t^s\langle\!\langle (Y).t[Y] \mid \mathtt{out}^{\mathtt{s}}(l,\mathtt{nl}(l,Y),t^s\langle\!\langle\dagger\rangle\!\rangle.\overline{k_t})\rangle\!\rangle. \tag{6}$$

for some fresh name k_t. Thus, the difference between process $\mathtt{extr}_{\mathtt{A}}(t,l)$ (given above) and process $\mathtt{extr}_{\mathtt{D}}(t,l)$ (given in (5)) is the third parameter, which enables us to have a controlled execution of adaptable processes, and a tight operational correspondence. We may now define:

$$\begin{aligned}
\mathtt{ts}_{\mathtt{A}}(t[P\,,Q]) &= 1+\mathtt{ts}_{\mathtt{A}}(P)\\
\mathtt{ts}((\nu a)P) &= \mathtt{ts}(P)\\
\mathtt{ts}(P\mid Q) &= \mathtt{ts}(P)+\mathtt{ts}(Q)\\
\mathtt{ts}(!P) = \mathtt{ts}(\mathbf{0}) &= \mathtt{ts}(\pi.P) = \mathtt{ts}(\langle P\rangle) = 0
\end{aligned}$$

Figure 6: Number of transactions.

DEFINITION 5.7. *Let ρ be a path. Also, let h_t, l_t, and p_ρ be fresh names. We define the translation $[\![\cdot]\!]^{\mathtt{A}}_{\rho} : \mathcal{C}_{\mathtt{A}} \to \mathcal{A}$ as*

$$\begin{aligned}
[\![t[P\,,Q]]\!]^{\mathtt{A}}_{\rho} &= t\big[[\![P]\!]^{\mathtt{A}}_{t,\rho}\big] \mid l_t.\,\big(\mathtt{extr}_{\mathtt{A}}(t,p_{t,\rho}) \mid p_\rho[[\![Q]\!]^{\mathtt{A}}_{\epsilon}]\big)\\
&\qquad \mid t.t^s\langle\!\langle (Y).t[Y] \mid \mathcal{T}_t(Y).\overline{h_t}\rangle\!\rangle\\
[\![\langle P\rangle]\!]^{\mathtt{A}}_{\rho} &= p_\rho\big[[\![P]\!]^{\mathtt{A}}_{\epsilon}\big]\\
[\![t.P]\!]^{\mathtt{A}}_{\rho} &= t.\overline{h_t}.[\![P]\!]^{\mathtt{A}}_{\rho}\\
[\![\,\overline{t}.P\,]\!]^{\mathtt{A}}_{\rho} &= \overline{t}.h_t.[\![P]\!]^{\mathtt{A}}_{\rho}
\end{aligned}$$

and as a homomorphism for the other operators.

As in the translation of $\mathcal{C}_{\mathtt{D}}$, the structure of a transaction and the number of its top-level processes dynamically changes if there is an abortion signal; we need need first to substitute Y in activation process $\mathcal{T}_t(Y)$ with the content of the location t. Also, whenever we need to extract processes located at $p_{t,\rho}$ we will substitute Y in process `out` (cf. (4)) by the content of the location t and count the current number of locations $p_{t,\rho}$, using function `nl`.

5.2.2 Translation Correctness for $\mathcal{C}_{\mathtt{A}}$

Compositionality for $[\![\cdot]\!]^{\mathtt{A}}_{\rho}$ follows directly from its definition:

THEOREM 5.8 (COMPOSITIONALITY). *The translation $[\![\cdot]\!]^{\mathtt{A}}_{\rho}$ is compositional, in the sense of Def. 4.1(1).*

The analysis of operational correspondence follows the same ideas as in the translation for discarding semantics. To precisely describe the number of required reduction steps, we introduce some auxiliary notions. Given a process P, the number of transaction scopes in a process P, denoted $\mathtt{ts}_{\mathtt{A}}(P)$ is as in Figure 6; the number of protected blocks $\mathtt{pb}_{\mathtt{A}}(P)$ is as in Figure 5. Also, function $\mathtt{d}(P)$ gives the set of default activities of nested transactions in P:

$$\mathtt{d}(t[P\,,Q]) = \{P\}\cup\mathtt{d}(P) \qquad \mathtt{d}(P\mid Q) = \mathtt{d}(P)\cup\mathtt{d}(Q)$$
$$\mathtt{d}(\mathbf{0}) = \mathtt{d}(\pi.P) = \mathtt{d}(!P) = \mathtt{d}(\langle P\rangle) = \emptyset \quad \mathtt{d}((\nu a)P) = \mathtt{d}(P)$$

and we define process $\mathtt{S}(P)$ as follows:

$$\mathtt{S}(P) = \begin{cases} \mathtt{pb}_{\mathtt{A}}(P) & \text{if } \mathtt{d}(P)=\emptyset\\ \mathtt{pb}_{\mathtt{A}}(P)+\sum_{i=1}^{n}\mathtt{pb}_{\mathtt{A}}(P_i) & \text{if } \mathtt{d}(P)=\{P_1,\ldots,P_n\}.\end{cases}$$

We then have the following result:

THEOREM 5.9 (OPERATIONAL CORRESPONDENCE). *Let P be a well-formed process in $\mathcal{C}_{\mathtt{A}}$. We have:*

1. *If $P \xrightarrow{\tau_a}_{\mathtt{A}} P'$ then $[\![P]\!]^{\mathtt{A}}_{\epsilon} \to^2 [\![P']\!]^{\mathtt{A}}_{\epsilon}$ for some a.*
2. *If $P \xrightarrow{\tau}_{\mathtt{A}} P'$ then $[\![P]\!]^{\mathtt{A}}_{\epsilon} \to^k [\![P']\!]^{\mathtt{A}}_{\epsilon}$ where either*
   (a) $P \equiv E[C[t[P_1\,,Q]] \mid D[\overline{t}.P_2]]$ and $k = 7+4\,\mathtt{ts}_{\mathtt{A}}(P_1)+\mathtt{S}(P_1)$
   (b) $P \equiv C[t[D[\overline{t}.P_1]\,,Q]]$ and $k = 7+4\,\mathtt{ts}_{\mathtt{A}}(D[P_1])+\mathtt{S}(D[P_1])$

   for some contexts C, D, E, processes P_1, Q, P_2, name t.
3. *If $[\![P]\!]^{\mathtt{A}}_{\epsilon} \to^* R$ then there is P' such that $P \xrightarrow{\tau}{}^*_{\mathtt{A}} P'$ and $R \to^* [\![P']\!]^{\mathtt{A}}_{\epsilon}$.*

We close this section by discussing a couple of examples:

EXAMPLE 5.10. *We apply the encoding to process P in Example 5.4. By the LTS of Fig.3, we have:*

$$P \xrightarrow{\tau}_{\mathtt{A}} s[\langle a\rangle \mid \langle b\rangle \mid \langle d\rangle\,,0] \mid \bar{s} \xrightarrow{\tau}_{\mathtt{A}} \langle a\rangle \mid \langle b\rangle \mid \langle d\rangle.$$

By expanding Def. 5.7, process $[\![P]\!]^{\mathtt{A}}_{s}$ is as follows. We write $\tilde{i}$ to abbreviate $i.\overline{h_i}$, with $i \in \{a,b,c,d\}$.

$$\begin{aligned}
& s\Big[t\Big[p_{t,s}\big[\tilde{a}\big] \mid p_{t,s}\big[\tilde{b}\big] \mid \tilde{c}\Big] \mid l_t.\big(\mathtt{extr}_{\mathtt{A}}(t,p_{t,s}) \mid p_s\big[\tilde{d}\big]\\
&\qquad \mid t.t^s\langle\!\langle (Y).t[Y] \mid \mathcal{T}_t(Y).\overline{h_t}\rangle\!\rangle\big)\Big] \mid l_s.\big(\mathtt{extr}_{\mathtt{A}}(s,p_s)\big)\\
&\quad \mid s.s^s\langle\!\langle (Y).s[Y] \mid \mathcal{T}_s(Y).\overline{h_s}\rangle\!\rangle \mid \overline{t}.h_t.\overline{s}.h_s\\
\to^2\ & s\Big[t\Big[p_{t,s}\big[\tilde{a}\big] \mid p_{t,s}\big[\tilde{b}\big] \mid \tilde{c}\Big] \mid l_t.\big(\mathtt{extr}_{\mathtt{A}}(t,p_{t,s}) \mid p_s\big[\tilde{d}\big]\big)\Big]\\
&\qquad \mid \overline{l_t}.k_t.\overline{h_t} \mid l_s.\big(\mathtt{extr}_{\mathtt{A}}(s,p_s)\big)\\
&\quad \mid s.s^s\langle\!\langle (Y).s[Y] \mid \mathcal{T}_s(Y).\overline{h_s}\rangle\!\rangle \mid h_t.\overline{s}.h_s\\
\to^7\ & s\Big[p_s\big[\tilde{a}\big] \mid p_s\big[\tilde{b}\big] \mid p_s\big[\tilde{d}\big]\Big] \mid l_s.\big(\mathtt{extr}_{\mathtt{A}}(s,p_s)\big)\\
&\qquad \mid s.s^s\langle\!\langle (Y).s[Y] \mid \overline{\mathcal{T}_s(Y).\overline{h_s}}\rangle\!\rangle \mid \overline{s}.h_s\\
\to^{10}\ & p_\epsilon\big[\tilde{a}\big] \mid p_\epsilon\big[\tilde{b}\big] \mid p_\epsilon\big[\tilde{d}\big] = [\![\langle a\rangle]\!]^{\mathtt{A}}_{\epsilon} \mid [\![\langle b\rangle]\!]^{\mathtt{A}}_{\epsilon} \mid [\![\langle d\rangle]\!]^{\mathtt{A}}_{\epsilon}.
\end{aligned}$$

EXAMPLE 5.11. *We apply the encoding on the Hotel booking scenario (§ 2) where the client cancels a reservation after booking and paying. Below Γ_{t_1} stands for $t_1.t_1^s\langle\!\langle (Y).t_1[Y] \mid \mathcal{T}_{t_1}(Y).\overline{h_{t_1}}\rangle\!\rangle$:*

$$\begin{aligned}
[\![R]\!]^{\mathtt{A}}_{\epsilon} \;=\;& t\big[book.\overline{h_b}.pay.\overline{h_p}.invoice.\overline{h_i}\\
&\quad \mid t_1[p_{t_1,t}[[\![DB]\!]^{\mathtt{A}}_{\epsilon}]] \mid l_{t_1}.(\mathtt{extr}_{\mathtt{A}}(t_1,p_{t_1,t})) \mid \Gamma_{t_1}\big]\\
&\quad \mid l_t.\,\big(\mathtt{extr}_{\mathtt{A}}(t,p_t) \mid p_t[\overline{refund}.h_r]\big)\\
&\quad \mid t.t^s\langle\!\langle (Y).t[Y] \mid \mathcal{T}_t(Y).\overline{h_t}\rangle\!\rangle\\
&\quad \mid \overline{book}.h_b.\overline{pay}.h_p.\overline{t}.h_t.refund.\overline{h_r}\\
\to^6\ & t\big[invoice.\overline{h_i} \mid t_1[p_{t_1,t}[[\![DB]\!]^{\mathtt{A}}_{\epsilon}]] \mid l_{t_1}.(t_1^s\langle\!\langle (Y).\\
&\quad \mid t_1[Y] \mid \mathtt{out}^{\mathtt{s}}(p_{t_1,t},\mathtt{nl}(p_{t_1,t},Y),t_1^s\langle\!\langle\dagger\rangle\!\rangle.\overline{k_{t_1}})\rangle\!\rangle)\\
&\quad \mid \Gamma_{t_1}\big] \mid l_t.(t^s\langle\!\langle (Y).t[Y]\\
&\quad \mid \mathtt{out}^{\mathtt{s}}(p_t,\mathtt{nl}(p_t,Y),t^s\langle\!\langle\dagger\rangle\!\rangle.\overline{k_t})\rangle\!\rangle \mid p_t[\overline{refund}.h_r])\\
&\quad \mid \overline{l_{t_1}}.k_{t_1}.\overline{l_t}.k_t.\overline{h_t} \mid h_t.refund.\overline{h_r}\\
\to^8\ & t\big[invoice.\overline{h_i} \mid \Gamma_{t_1} \mid p_t[[\![DB]\!]^{\mathtt{A}}_{\epsilon}]\big]\\
&\quad \mid \mathtt{out}^{\mathtt{s}}(p_t,1,t^s\langle\!\langle\dagger\rangle\!\rangle.\overline{k_t}) \mid p_t[\overline{refund}.h_r]\\
&\quad \mid k_t.\overline{h_t} \mid h_t.refund.\overline{h_r}\\
\to^6\ & p_t[[\![DB]\!]^{\mathtt{A}}_{\epsilon}] = [\![\langle DB\rangle]\!]^{\mathtt{A}}_{\epsilon}.
\end{aligned}$$

When the client cancels the reservation, it first synchronizes on name t. The mechanism for compensation is enclosed in location $p_t[\overline{refund}.h_t]$; it allows the client to claim a refund. The release of this mechanism from transaction t will be activated via a synchronization on name l_t. But first the encoding of the protected block $\langle DB\rangle$ from transaction t_1 has to be moved out of t. Subsequently, synchronization on names l_1 and $\overline{l_{t_1}}$ takes place. Ultimately, we get $[\![R]\!]^{\mathtt{A}}_{\epsilon} \to^{20} p_t[[\![DB]\!]^{\mathtt{A}}_{\epsilon}]$.

The encoding of R under a discarding semantics requires less steps: $[\![R]\!]^{\mathtt{D}}_{\epsilon} \to^{10} p_t[[\![\mathbf{0}]\!]^{\mathtt{D}}_{\epsilon}]$. This is expected, given the different levels of protection that these two semantics offer.

6. SUBJECTIVE VS OBJECTIVE UPDATES

We substantiate our claim on the *efficiency* that a language with subjective update prefixes offers with respect to objective update prefixes, denoted $l^o\{(X).P\}$. We consider efficiency in terms of reduction steps, so we recall the reduction rule considered in [7]:

$$E\Big[C\big[l[P]\big] \mid D\big[l^o\{(X).Q\}.R\big]\Big] \to E\Big[C\big[Q\{^P/_X\}\big] \mid D\big[R\big]\Big]$$

It turns out that the main (and only) difference between our encodings and the encodings in [7] is in the process $\mathtt{out}^{\mathtt{s}}(p_{t,\rho}, n, Q)$, which is part of $\mathtt{extr}(t, p_{t,\rho})$ — see (4) and (5). Indeed, process $\mathtt{out}^{\mathtt{s}}(p_{t,\rho}, n, Q)$ extracts n processes located at $p_{t,\rho}$.

The analog of $\mathtt{out}^{\mathtt{s}}(p_{t,\rho}, n, Q)$ in [7] with objective update is as follows. If $n = 0$ then $\mathtt{out}^{\mathtt{o}}(p_{t,\rho}, 0, Q) = Q$; otherwise we have

$$\mathtt{out}^{\mathtt{o}}(p_{t,\rho}, n, Q) = p^{o}_{t,\rho}\big\{(X_1, \ldots, X_n).\ z^{o}\big\{\prod_{i=1}^{n} p_\rho[X_i] \mid Q\big\}\big\}.z[\mathbf{0}]$$

Thus, $\mathtt{out}^{\mathtt{s}}(p_{t,\rho}, n, Q)$ and $\mathtt{out}^{\mathtt{o}}(p_{t,\rho}, n, Q)$ above differ in the use of an additional synchronization on name z. This appears indispensable: under a semantics with objective update, after n updates, processes located at p_ρ will stay at location t. To avoid leaving such processes in the wrong location, the encodings in [7] use an (objective) update on auxiliary location z, so to take them out of t once n updates on $p_{t,\rho}$ have been executed. When moving from objective to subjective updates this synchronization on z is no longer needed — clearly, the improvement will be proportional to the number of compensation operations in the source language (here $\mathcal{C}_{\mathtt{D}}$ and $\mathcal{C}_{\mathtt{A}}$).

We may then conclude that our encodings of compensable processes into adaptable processes with subjective update are more efficient than the encodings with objective update [7]: for both discarding and aborting semantics, our encodings require one less reduction step to mimic a compensation step in the source language.

7. CONCLUSIONS

The quest for programming abstractions that suit emerging computational settings such as CAS is a multi-faceted problem. Rather than developing new languages from scratch, one approach is to build on languages already developed for mobile, autonomic, and service-oriented computing. Here we formally connect programming abstractions for compensation handling (typical of models for services and long-running transactions) and for run-time adaptation. We improve our results in [7] by offering more efficient encodings that exploit a novel programming abstraction based on *subjective* process mobility.

Our work uncovers an interesting dichotomy: should one appeal to objective or to subjective updates? In this paper, we have considered the calculus of adaptable processes of [2] with subjective update prefixes only. This is because our goal was to understand the effect of subjective mobility in the efficiency of encodings of compensable processes. While subjective updates appear *more autonomous* (i.e., determined by a located process itself, not by its environment) than the objective updates of [7], we believe that the choice of objective and subjective update largely depends on the application at hand: there are practical instances of dynamic reconfiguration for which each form of update is better suited. Hence, a general specification language should probably include both objective and subjective updates.

We notice that subjective updates can represent objective updates, at least in an ad-hoc manner. Consider processes S and S':

$$\begin{aligned} S &= C_1\big[l[P] \mid R_1\big] \mid C_2\big[l^{o}\{(X).Q\}.R_2\big] \\ S' &= C_1\big[l[P] \mid l_1^{s}\langle\!\langle (X).X \rangle\!\rangle \mid R_1\big] \mid C_2\big[l^{s}\langle\!\langle (X).l_1[Q] \rangle\!\rangle.R_2\big] \end{aligned}$$

Intuitively, S' is a subjective variant of S: using two reductions, S' emulates the movement induced by objective updates in S (see (1)):

$$\begin{aligned} S' &\to C_1\big[l_1^{s}\langle\!\langle (X).X \rangle\!\rangle \mid R_1\big] \mid C_2\big[l_1[Q\{^{P}/_{X}\}].R_2\big] \\ &\to C_1\big[Q\{^{P}/_{X}\} \mid R_1\big] \mid C_2\big[R_2\big] \end{aligned}$$

A similar (ad-hoc) transformation can be defined to represent subjective updates using objective ones. In both cases, however, the ability of emulating a certain direction of process movement comes at the price of additional reduction steps that induce inefficient representations in the long run. This reinforces our claim that both forms of update should be kept natively in a specification language.

In future work, we would like to study the dichotomy between subjective and objective updates in the context of other formal languages for CAS, such as SCEL [6], AbC [1], and the calculus with code mobility in [5]. One first challenge is that dynamic update as studied here is based in the point-to-point communication discipline of CCS and the π-calculus; in contrast, communication in SCEL and AbC is *attribute-based*, and permits interactions between with groups of partners as selected by appropriate predicates.

Acknowledgments.

We are grateful to the anonymous reviewers for their remarks and useful suggestions. This research has been partially supported by EU COST Actions IC1201 (BETTY), IC1402 (ARVI), and IC1405 (Reversible Computation), by CNRS PICS project 07313 (SuCCeSS), and by grant ON174026 of the Ministry of Education and Science, Serbia. Pérez is also affiliated to NOVA Laboratory for Computer Science and Informatics (NOVA LINCS), Universidade Nova de Lisboa, Portugal (Ref. UID/CEC/04516/2013).

8. REFERENCES

[1] Y. A. Alrahman, R. De Nicola, and M. Loreti. On the power of attribute-based communication. In *Proc. of FORTE 2016*, volume 9688 of *LNCS*, pages 1–18. Springer, 2016.

[2] M. Bravetti, C. D. Giusto, J. A. Pérez, and G. Zavattaro. Adaptable processes. *Logical Methods in Computer Science*, 8(4), 2012.

[3] L. Cardelli and A. D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.

[4] G. Castagna, J. Vitek, and F. Z. Nardelli. The seal calculus. *Inf. Comput.*, 201(1):1–54, 2005.

[5] F. Damiani, M. Viroli, D. Pianini, and J. Beal. Code mobility meets self-organisation: A higher-order calculus of computational fields. In *Proc. of FORTE 2015*, volume 9039 of *LNCS*, pages 113–128. Springer, 2015.

[6] R. De Nicola, M. Loreti, R. Pugliese, and F. Tiezzi. A formal approach to autonomic systems programming: The SCEL language. *TAAS*, 9(2):7:1–7:29, 2014.

[7] J. Dedeić, J. Pantović, and J. A. Pérez. On compensation primitives as adaptable processes. In *EXPRESS/SOS 2015*, volume 190 of *EPTCS*, pages 16–30, 2015.

[8] D. Gorla. Towards a unified approach to encodability and separation results for process calculi. *Inf. Comput.*, 208(9):1031–1053, 2010.

[9] I. Lanese, C. Vaz, and C. Ferreira. On the expressive power of primitives for compensation handling. In *Proc. of ESOP 2010*, volume 6012 of *LNCS*, pages 366–386. Springer, 2010.

[10] I. Lanese and G. Zavattaro. Decidability results for dynamic installation of compensation handlers. In *COORDINATION*, volume 7890 of *LNCS*, pages 136–150. Springer, 2013.

[11] R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[12] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

[13] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher Order Paradigms*. PhD thesis, University of Edinburgh, 1992.