Identifying Commonalities and Differences in Object Life Cycles using Behavioral Inheritance

W.M.P. van der Aalst[1] and T. Basten[2]

[1] Dept. of Technology Management and Dept. of Computing Science, Eindhoven University of Technology, The Netherlands, w.m.p.v.d.aalst@tue.nl
[2] Dept. of Electrical Engineering, Eindhoven University of Technology, The Netherlands, a.a.basten@tue.nl

Abstract. The behavioral-inheritance relations of [7, 8] can be used to compare the life cycles of objects defined in terms of Petri nets. They yield partial orders on object life cycles (OLCs). Based on these orders, we define concepts such as the greatest common divisor and the least common multiple of a set of OLCs. These concepts have practical relevance: In component-based design, workflow management, ERP reference models, and electronic-trade procedures, there is a constant need for identifying commonalities and differences in OLCs. Our results provide the theoretical basis for comparing, customizing, and unifying OLCs.

Key words: Petri nets, inheritance, lattices, partial orders, object-oriented methods, workflow management.

1 Introduction

For several years, we have been working on notions of *inheritance of behavior* [7, 8]. Inheritance is a key issue in object-oriented design [10]. It allows for the definition of subclasses that inherit features of some superclass. Inheritance is well defined for static properties of classes such as attributes and methods. However, there is no general agreement on the meaning of inheritance when considering the dynamic behavior of objects, captured by their life cycles. In our work, we use Petri nets [18, 19] for defining object life cycles (OLCs); they allow for a graphical representation with an explicit representation of object states. In [7, 8], four behavioral-inheritance notions have been defined, based on the principle that by blocking and/or hiding methods of a subclass the resulting behavior should match the behavior of the superclass.

We have applied the behavioral-inheritance concepts in different domains ranging from workflow management [4] and electronic commerce [3] to object-oriented methods [7, 8] and component-based software architectures [5]. In each of these applications, objects are designed and compared. The objects of interest can be insurance claims, orders, bank accounts, hardware modules, or software components. One thing they have in common is that they have a *life cycle*. The inheritance notions have been used as a basis for the comparison of these OLCs. The applications revealed a new and intriguing question: Given a set of OLCs, what do these OLCs have in common? In this paper, we provide some fundamental results that can be used to answer this question.

Consider a set of OLCs that are variants of some process (a workflow or trade procedure, or the control flow in a hardware or software component). Each of the inheritance relations yields a partial order that can be used to reason about a common super- or subclass of the variants. The *Greatest Common Divisor* (GCD) is the common superclass which preserves as much information about the OLCs as possible, i.e., it is not possible to construct a more detailed OLC which is also a superclass of all variants. The GCD describes the behavior all variants agree on. The *Least Common Multiple* (LCM) is the most compact OLC which is still a subclass of all variants. For each of the application domains mentioned, there is a clear use for such concepts. Consider for example two similar software components. The GCD can be used to deduce what both components have in common; the LCM can be used to construct a generic component which can be used to replace the other two. Another example is the use of ad-hoc workflows. Workflow management systems such as InConcert (TIBCO) allow for case-specific variants of a workflow process. Both the GCD and the LCM of these variants can be used to generate meaningful management information [4].

In this paper, we define the concepts of GCDs and LCMs based on the four inheritance relations mentioned earlier. Since none of them forms a (complete) lattice, a restrictive definition leads to situations where there is no GCD (LCM) and a more liberal definition leads to situations where there are multiple GCDs (LCMs). Both situations are undesirable. We tackle this problem by giving both more restrictive and more liberal definitions. For the latter, we use the terms *Maximal Common Divisor* (MCD) and *Minimal Common Multiple* (MCM). We use the Dedekind-MacNeille [17] completion to turn an inheritance partial order into a complete lattice with virtual nodes. In such a lattice, each set of variants has a GCD and an LCM. However, they may correspond to a so-called virtual OLC. Although a virtual OLC cannot be represented by a single Petri net, it provides meaningful information on commonalities and differences.

The paper is organized as follows. Section 2 introduces preliminaries. The behavioral-inheritance concepts are given in Section 3. The other sections deal with GCDs, LCMs, MCDs, and MCMs. Section 4 studies these notions in the context of life-cycle inheritance, the most general form of inheritance. In Section 5, the results are extended to the three other notions of inheritance. Section 6 uses the Dedekind-MacNeille completion to guarantee the existence of GCDs and LCMs. We conclude with some remarks on the application of our results.

2 Preliminaries

This section introduces the techniques used in the remainder. Standard definitions for Petri nets are given. Moreover, more advanced concepts such as branching bisimilarity and OLCs are presented.

2.1 Labeled Place/Transition nets

We define a variant of the classic Petri-net model, namely labeled Place/Transition nets. For an elaborate introduction to Petri nets, the reader is referred to [12, 18, 19]. Let L be some set of *action labels*. These labels correspond to methods when modeling OLCs.

Definition 2.1. (Labeled P/T-nets)[1] An L-labeled Place/Transition net, or simply labeled P/T-net, is a tuple (P, T, F, ℓ) where:

1. P is a finite set of *places*,
2. T is a finite set of *transitions* such that $P \cap T = \emptyset$,
3. $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the *flow relation*, and
4. $\ell : T \to L$ is a *labeling function*.

A *marked*, L-labeled P/T-net is a pair (N, s), where $N = (P, T, F, \ell)$ is an L-labeled P/T-net and where s is a bag over P denoting the *marking* of the net. The set of all marked, L-labeled P/T-nets is denoted $\mathcal{N}$.

A marking is a *bag* over the set of places P, i.e., it is a function from P to the natural numbers. We use square brackets for the enumeration of a bag, e.g., $[a^2, b, c^3]$ denotes the bag with two a-s, one b, and three c-s. The sum of two bags $(X + Y)$, the difference $(X - Y)$, the presence of an element in a bag $(a \in X)$, and the notion of subbags $(X \leq Y)$ are defined in a straightforward way and they can handle a mixture of sets and bags.

Transition labeling is needed for two reasons. First, a P/T-net modeling an OLC may contain several transitions referring to a single method (identified by the label) in the OLC. Second, we use transition labels as a mechanism to abstract from (internal) methods. For simplicity, we assume that transition labels are identical to transition identifiers unless explicitly stated otherwise.

Let $N = (P, T, F, \ell)$ be a labeled P/T-net. Elements of $P \cup T$ are called *nodes*. A node x is an *input node* of another node y iff there is a directed arc from x to y (i.e., xFy). Node x is an *output node* of y iff yFx. For any $x \in P \cup T$, $\overset{N}{\bullet} x = \{y \mid yFx\}$ and $x \overset{N}{\bullet} = \{y \mid xFy\}$; the superscript N may be omitted if clear from the context.

The dynamic behavior of marked, labeled P/T-nets is defined by a *firing rule*.

Definition 2.2. (Firing rule) Let $(N = (P, T, F, \ell), s)$ be a marked, labeled P/T-net. Transition $t \in T$ is *enabled*, denoted $(N, s)[t\rangle$, iff $\bullet t \leq s$. The *firing rule* $\_[\_\rangle\_ \subseteq \mathcal{N} \times L \times \mathcal{N}$ is the smallest relation satisfying for any $(N = (P, T, F, \ell), s) \in \mathcal{N}$ and any $t \in T$, $(N, s)[t\rangle \Rightarrow (N, s)\ [\ell(t)\rangle\ (N, s - \bullet t + t\bullet)$.

A transition firing is also referred to as an *action*.

Definition 2.3. (Reachable markings) Let (N, s_0) be a marked, labeled P/T-net in $\mathcal{N}$. A marking s is *reachable* from the initial marking s_0 iff there exists a sequence of enabled transitions whose firing leads from s_0 to s. The set of reachable markings of (N, s_0) is denoted $[N, s_0\rangle$.

Definition 2.4. (Connectedness) A net $N = (P, T, F, \ell)$ is *weakly connected*, or simply *connected*, iff, for every two nodes x and y in $P \cup T$, $x(F \cup F^{-1})^* y$, where R^{-1} is the inverse and R^* the reflexive and transitive closure of a relation R. Net N is *strongly connected* iff, for every two nodes x and y, xF^*y.

[1] In the literature, the class of Petri nets introduced in Definition 2.1 is sometimes referred to as the class of (labeled) *ordinary* P/T-nets to distinguish it from the class of Petri nets that allows more than one arc between a place and a transition.

We assume that all nets are weakly connected and have at least two nodes.

Definition 2.5. (Boundedness, safeness) A marked net $(N = (P, T, F, \ell), s)$ is *bounded* iff the set of reachable markings $[N, s\rangle$ is finite. It is *safe* iff, for any $s' \in [N, s\rangle$ and any $p \in P$, $s'(p) \leq 1$. Note that safeness implies boundedness.

Definition 2.6. (Dead transitions, liveness) Let $(N = (P, T, F, \ell), s)$ be a marked, labeled P/T-net. A transition $t \in T$ is *dead* in (N, s) iff there is no reachable marking $s' \in [N, s\rangle$ such that $(N, s')[t\rangle$. (N, s) is *live* iff, for every reachable marking $s' \in [N, s\rangle$ and $t \in T$, there is a reachable marking $s'' \in [N, s'\rangle$ such that $(N, s'')[t\rangle$. Note that liveness implies the absence of dead transitions.

2.2 Branching bisimilarity

To formalize the inheritance concepts in this paper, we need a notion of equivalence. We choose *branching bisimilarity* [13] as the standard equivalence relation. The notion of a *silent action* is pivotal to branching bisimilarity. Silent actions, denoted with the label τ, are actions that cannot be observed. Thus, only the firings of transitions of a P/T-net with a label different from τ are observable.

We distinguish *successful termination* from *deadlock*. A *termination predicate* $\downarrow \subseteq \mathcal{N}$ defines in what states a marked net can terminate successfully. A marked net that cannot perform any actions or terminate successfully is in a *deadlock*.

We need two auxiliary definitions: (1) a relation expressing that a marked net can evolve via zero or more τ actions into another marked net; (2) a predicate expressing that a marked net can terminate via zero or more τ actions.

Definition 2.7. The relation $\_ \Longrightarrow \_ \subseteq \mathcal{N} \times \mathcal{N}$ is defined as the smallest relation satisfying, for any $p, p', p'' \in \mathcal{N}$, $p \Longrightarrow p$ and $(p \Longrightarrow p' \wedge p' \, [\tau\rangle \, p'') \;\Rightarrow\; p \Longrightarrow p''$. The predicate $\Downarrow \_ \subseteq \mathcal{N}$ is defined as the smallest set of marked, labeled P/T-nets satisfying, for any $p, p' \in \mathcal{N}$, $\downarrow p \Rightarrow \Downarrow p$ and $(\Downarrow p \wedge p' \, [\tau\rangle \, p) \;\Rightarrow\; \Downarrow p'$.

For any two marked, L-labeled P/T-nets $p, p' \in \mathcal{N}$ and action $\alpha \in L$, $p \, [(\alpha)\rangle \, p'$ is an abbreviation of $(\alpha = \tau \wedge p = p') \vee p \, [\alpha\rangle \, p'$. Thus, $p \, [(\tau)\rangle \, p'$ means that zero τ actions are performed, when the first disjunct is satisfied, or that one τ action is performed, when the second disjunct is satisfied. For any observable action $a \in L \setminus \{\tau\}$, the first disjunct of the predicate can never be satisfied. Hence, $p \, [(a)\rangle \, p'$ simply equals $p \, [a\rangle \, p'$, meaning that a single a action is performed.

Definition 2.8. (Branching bisimilarity) A binary relation $\mathcal{R} \subseteq \mathcal{N} \times \mathcal{N}$ is called a *branching bisimulation* if and only if, for any $p, p', q, q' \in \mathcal{N}$ and $\alpha \in L$,

1. $p\mathcal{R}q \wedge p \, [\alpha\rangle \, p' \Rightarrow (\exists \, q', q'' : q', q'' \in \mathcal{N} : q \Longrightarrow q'' \wedge q'' \, [(\alpha)\rangle \, q' \wedge p\mathcal{R}q'' \wedge p'\mathcal{R}q')$,
2. $p\mathcal{R}q \wedge q \, [\alpha\rangle \, q' \Rightarrow (\exists \, p', p'' : p', p'' \in \mathcal{N} : p \Longrightarrow p'' \wedge p'' \, [(\alpha)\rangle \, p' \wedge p''\mathcal{R}q \wedge p'\mathcal{R}q')$,
3. $p\mathcal{R}q \Rightarrow (\downarrow p \Rightarrow \Downarrow q \wedge \downarrow q \Rightarrow \Downarrow p)$.

Two marked, labeled P/T-nets are called *branching bisimilar*, denoted $p \sim_b q$, if and only if there exists a branching bisimulation $\mathcal{R}$ such that $p\mathcal{R}q$.

Branching bisimilarity is an equivalence relation on $\mathcal{N}$, i.e., $\sim_b$ is reflexive, symmetric, and transitive (see [7] for a detailed proof).

2.3 Object life cycles

Petri nets allow for the graphical representation of OLCs with an explicit representation of states and a clear definition of the initial (object creation) and final (object termination) state. OLCs correspond to the diagrams used in object-oriented methods (e.g., statechart diagrams in UML [10]), process definitions used by workflow management systems [14, 4], reference models used in ERP systems (e.g., EPCs used by SAP [15]), and trade procedures as defined in [16].

Definition 2.9. (Object life cycle) Let $N = (P, T, F, \ell)$ be an L-labeled P/T-net and $\bar{t}$ a fresh identifier not in $P \cup T$. N is an *object life cycle* (OLC) iff:

1. *object creation*: P contains an input place i such that $\bullet i = \emptyset$,
2. *object completion*: P contains an output place o such that $o\bullet = \emptyset$,
3. *connectedness*: $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\}, \ell \cup \{(\bar{t}, \tau)\})$ is strongly connected,
4. *safeness*: $(N, [i])$ is safe,
5. *proper completion*: for any marking $s \in [N, [i]\rangle$, $o \in s$ implies $s = [o]$,
6. *option to complete*: for any marking $s \in [N, [i]\rangle$, $[o] \in [N, s\rangle$, and
7. *absence of dead methods*: $(N, [i])$ contains no dead transitions.

The set of all OLCs is denoted $\mathcal{O}$.

An OLC satisfies seven requirements. First, an OLC has one place i without any input transitions. A token in i corresponds to an object which is created, i.e., at the beginning of its life cycle. Second, an OLC has one place o without output transitions. A token in o corresponds to an object that is destroyed. Third, an OLC should not have "dangling" transitions and/or places. Thus, every node of an OLC should be located on a path from i to o. This requirement corresponds to strongly connectedness if o is connected to i via an additional transition $\bar{t}$. The net $\bar{N}$ used to formulate the connectedness constraint is called the *short-circuited* net. The label of the new transition is not important and simply set to τ. The fourth requirement says that an OLC is safe. This is a reasonable assumption since places in an OLC correspond to conditions which are either true (marked by a token) or false (empty). The fifth requirement states that the moment a token is put into o all the other places should be empty, which corresponds to the completion of an OLC without leaving dangling references. The sixth requirement states that starting from the initial marking $[i]$ it is always possible to reach the marking with one token in o, which means that it is always feasible to complete the OLC. The last requirement implies that for each transition there is a scenario in which the transition is performed.

The notion of an OLC is strongly related to the notion of a sound workflow net [2, 4]. A workflow net also describes the life cycle of one object, often called a *case* or a *workflow instance*. The applicability of the results in this paper transcends workflow management. Therefore, we prefer the term object life cycle (OLC). Since transition labels in OLCs correspond to methods, we also use the term "method" implicitly for transitions.

The last four requirements in Definition 2.9 coincide with liveness and safeness of the short-circuited net [1]. Thus, we can use standard techniques for checking the life-cycle requirements. Our tool *Woflan* [20] has been specifically designed to analyze the requirements stated in Definition 2.9.

We introduced branching bisimilarity as the standard equivalence. Recall that branching bisimilarity distinguishes successful termination and deadlock. An OLC can only terminate successfully in marking $[o]$.

Definition 2.10. The class of marked, labeled P/T-nets $\mathcal{N}$ is equipped with the following termination predicate: $\downarrow = \{(N, [o]) \mid N \in \mathcal{O}\ \}$.

Definition 2.11. (Behavioral equivalence of OLCs) For OLCs N_0 and N_1 in $\mathcal{O}$, $N_0 \cong N_1$ if and only if $(N_0, [i]) \sim_b (N_1, [i])$.

The set of *visible* actions that an OLC can perform is called the *alphabet* of the OLC. Since an OLC does not have any dead transitions, its alphabet simply is the set of its transition labels excluding silent action τ.

Definition 2.12. (Alphabet) For any OLC $N = (P, T, F, \ell)$ in $\mathcal{O}$, its alphabet $\alpha(N)$ equals $\{\ell(t) \mid t \in T \wedge \ell(t) \neq \tau\}$.

3 Inheritance

In this section, we define four *behavioral*-inheritance relations for OLCs. For a detailed motivation and an overview of related work, we refer to [8]. Consider two OLCs x and y. When is x a subclass of y? Intuitively, one could say that x is a subclass of y iff x can do what y can do. Clearly, all methods of y should also be present in x. Moreover, x will typically add new methods. Therefore, it is reasonable to demand that x can do what y can do with respect to the methods present in y. With respect to new methods (i.e., methods present in x but not in y), there are basically two mechanisms which can be used. The first one simply disallows the execution of any new methods.

> *If it is not possible to distinguish the behaviors of x and y when only methods of x that are also present in y are executed, then x is a subclass of y.*

This definition conforms to *blocking* methods new in x. The resulting inheritance concept is called *protocol inheritance*; x inherits the protocol of y.

Another mechanism would be to allow for the execution of new methods but to consider only the effects of old ones.

> *If it is not possible to distinguish the behaviors of x and y when arbitrary methods of x are executed but when only the effects of methods that are also present in y are considered, then x is a subclass of y.*

This inheritance notion is called *projection inheritance*; it conforms to *hiding* methods new in x. This can be achieved by renaming these methods to the silent action τ.

Although the distinction between the two inheritance mechanisms may seem subtle, the corresponding inheritance notions are quite different. To illustrate

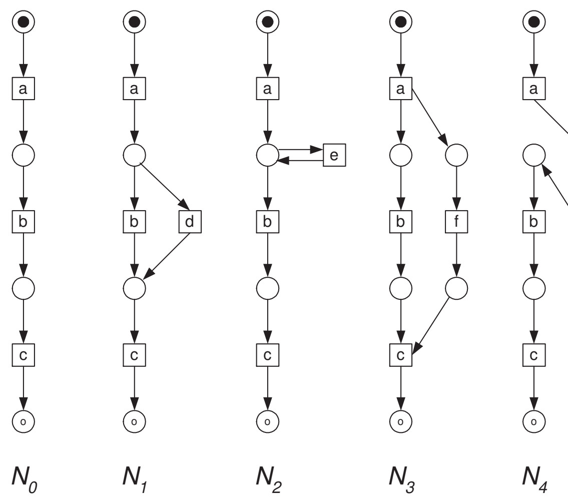

Fig. 1. Five object life cycles.

this, we use the five OLCs of Figure 1. N_0 corresponds to a sequential OLC consisting of three methods a, b, and c. Each of the other OLCs extends N_0 with one additional method. In N_1, method d can be executed instead of b. N_1 is a subclass of N_0 under protocol inheritance; if d is blocked, N_1 is equivalent to N_0. N_1 is not a subclass of N_0 under projection inheritance, because it is possible to skip method b by executing the (hidden) method d. In N_2, method e can be executed arbitrarily many times between a and b. N_2 is a subclass of N_0 under protocol inheritance; if e is blocked, then N_2 equals N_0. N_2 is also a subclass of N_0 under projection inheritance; if every execution of e is hidden, then N_2 is equivalent (as defined in Definition 2.11) to N_0. In OLC N_3, method f is executed in parallel with b. N_3 is not a subclass of N_0 under protocol inheritance; if f is blocked, then c cannot be executed. However, N_3 is a subclass of N_0 under projection inheritance. If one hides the newly-added method f, one cannot distinguish N_3 and N_0. Method g is inserted between a and b in the remaining OLC N_4. N_4 is not a subclass of N_0 under protocol inheritance; if g is blocked, the OLC deadlocks after executing a. However, N_4 is a subclass of N_0 under projection inheritance. If one hides g, one cannot observe any differences between the behaviors of N_4 and N_0.

The two mechanisms (i.e., blocking and hiding) result in orthogonal inheritance notions. We also consider combinations of the two. An OLC is a subclass of another OLC under *protocol/projection inheritance* iff both by hiding the new methods and by blocking the new methods one cannot detect any differences, i.e., it is a subclass under both protocol and projection inheritance. In Figure 1, N_2 is a subclass of N_0 under protocol/projection inheritance. The two mechanisms can also be used to obtain a more general form of inheritance. An OLC is a subclass of another OLC under *life-cycle inheritance* iff by blocking some newly-added methods and by hiding some others one cannot distinguish them. All OLCs in Figure 1 are subclasses of N_0 under life-cycle inheritance.

To formalize the inheritance relations, we define two operators on P/T-nets: *encapsulation* for blocking and *abstraction* for hiding methods. They are inspired by the encapsulation and abstraction operators from process algebra [6].[2]

Definition 3.1. (Encapsulation) Let $N = (P, T_0, F_0, \ell_0)$ be an L-labeled P/T-net. For any $H \subseteq L \setminus \{\tau\}$, the encapsulation operator ∂_H is a function that removes from a given P/T-net all transitions with a label in H. Formally, $\partial_H(N) = (P, T_1, F_1, \ell_1)$ such that $T_1 = \{t \in T_0 \mid \ell_0(t) \notin H\}$, $F_1 = F_0 \cap ((P \times T_1) \cup (T_1 \times P))$, and $\ell_1 = \ell_0 \cap (T_1 \times L)$.

Note that removing transitions from an OLC as defined in Definition 2.9 might yield a P/T-net that is no longer an OLC.

Definition 3.2. (Abstraction) Let $N = (P, T, F, \ell_0)$ be an L-labeled P/T-net. For any $I \subseteq L \setminus \{\tau\}$, the abstraction operator τ_I is a function that renames all transition labels in I to the silent action τ. Formally, $\tau_I(N) = (P, T, F, \ell_1)$ such that, for any $t \in T$, $\ell_0(t) \in I$ implies $\ell_1(t) = \tau$ and $\ell_0(t) \notin I$ implies $\ell_1(t) = \ell_0(t)$.

The formal definitions of the four inheritance relations are slightly more general than the informal definitions given above: An OLC is a subclass of another OLC if and only if there exists *some* set of methods such that encapsulating or hiding these methods in the first OLC yields the other OLC. Not requiring that the methods being encapsulated or hidden must be *exactly* the newly-added methods can sometimes be convenient. In [7, 8], it is shown that the formal and informal definitions are equivalent. Recall Definition 2.8 (branching bisimilarity, $\sim_b$).

Definition 3.3. (Inheritance relations)

1. *Protocol inheritance*: For any OLCs N_0 and N_1 in $\mathcal{O}$, OLC N_1 is a subclass of N_0 under protocol inheritance, denoted $N_1 \leq_{pt} N_0$, iff there is an $H \subseteq L \setminus \{\tau\}$ such that $(\partial_H(N_1), [i]) \sim_b (N_0, [i])$.
2. *Projection inheritance*: For any OLCs N_0 and N_1 in $\mathcal{O}$, OLC N_1 is a subclass of N_0 under projection inheritance, denoted $N_1 \leq_{pj} N_0$, iff there is an $I \subseteq L \setminus \{\tau\}$ such that $(\tau_I(N_1), [i]) \sim_b (N_0, [i])$.
3. *Protocol/projection inheritance*: For any OLCs N_0 and N_1 in $\mathcal{O}$, OLC N_1 is a subclass of N_0 under protocol/projection inheritance, denoted $N_1 \leq_{pp} N_0$, iff there is an $H \subseteq L \setminus \{\tau\}$ such that $(\partial_H(N_1), [i]) \sim_b (N_0, [i])$ *and* an $I \subseteq L \setminus \{\tau\}$ such that $(\tau_I(N_1), [i]) \sim_b (N_0, [i])$.
4. *Life-cycle inheritance*: For any OLCs N_0 and N_1 in $\mathcal{O}$, N_1 is a subclass of N_0 under life-cycle inheritance, denoted $N_1 \leq_{lc} N_0$, iff there are an $I \subseteq L \setminus \{\tau\}$ and an $H \subseteq L \setminus \{\tau\}$ such that $I \cap H = \emptyset$ and $(\tau_I \circ \partial_H(N_1), [i]) \sim_b (N_0, [i])$.

Note that for life-cycle inheritance the new methods are partitioned into two sets H and I: methods that are blocked by means of the operator ∂_H and methods that are hidden by means of τ_I. It is easy to see that protocol/projection inheritance implies both protocol and projection inheritance. Moreover, both protocol

[2] Note that the terms "abstraction" and "encapsulation" in process algebra have a different meaning than the same terms in object-oriented design. In this paper, they always refer to the process-algebraic concepts.

and projection inheritance imply life-cycle inheritance. However, life-cycle inheritance does not imply protocol or projection inheritance.

The inheritance relations have a number of desirable properties. First, they are preorders (i.e., they are reflexive and transitive; see Property 6.19 in [8]). Furthermore, if one OLC is a subclass of another OLC under any of the four inheritance relations and vice versa, then the two OLCs are equivalent as defined in Definition 2.11 (i.e., the two OLCs are branching bisimilar; see Property 6.21 in [8]). In other words, the four inheritance relations are anti-symmetric. A relation that is reflexive, anti-symmetric, and transitive is a partial order.

Property 3.4. Assuming $\cong$, as defined in Definition 2.11, as the equivalence on OLCs, $\leq_{lc}$, $\leq_{pt}$, $\leq_{pj}$, and $\leq_{pp}$ are partial orders.

4 GCDs and LCMs under life-cycle inheritance

Each of the four notions of inheritance provides a partial ordering on OLCs. This inspired us to investigate whether it is possible to define the notions of a *Greatest Common Divisor* (GCD) and a *Least Common Multiple* (LCM) for sets of OLCs. In this section, we restrict ourselves to life-cycle inheritance (Definition 3.3-4). In Section 5, we consider the other three inheritance notions. We use the term *variant* for an OLC in a set of OLCs. The idea is that the GCD should capture the commonality of the variants, i.e., the part where they agree on. The LCM should capture all possible behaviors of all the variants. Consider for example the five OLCs of Figure 1. The GCD of these OLCs should be N_0. All the OLCs execute a, b, and c in sequential order. Each of the five variants is a subclass of N_0 and it is not possible to find a different OLC that is also a superclass of N_0 through N_4 and at the same time a subclass of N_0. Figure 2 shows $N_{GCD} = N_0$ as the GCD of the five OLCs of Figure 1. It also shows the OLC N_{LCM}. N_{LCM} is a subclass of each of the five variants considered. Moreover, it is not possible to find a different OLC which is also a subclass of N_0 through N_4 and at the same time a superclass of N_{LCM}. Thus, N_{LCM} is a good choice for the LCM of N_0 through N_4. Any sequence of transition firings generated by one of the five OLCs can also be generated by N_{LCM} after the appropriate abstraction.

To formalize the GCD and LCM concepts, we need some partial-order theory.

Definition 4.1. (Lattices) Let $(Q, \leq)$ be a partial order; let $S \subseteq Q$ and $q \in Q$.

1. q is an *upper bound* of S iff $s \leq q$ for all $s \in S$;
2. $S^{\uparrow} = \{x \in Q \mid (\forall\, s : s \in S : s \leq x)\}$ is the set of all upper bounds of S;
3. q is a *lower bound* of S iff $q \leq s$ for all $s \in S$;
4. $S^{\downarrow} = \{x \in Q \mid (\forall\, s : s \in S : x \leq s)\}$ is the set of all lower bounds of S;
5. q is the *least upper bound* (lub) of S iff q is an upper bound of S and $q \leq s$ for all $s \in S^{\uparrow}$;
6. q is the *greatest lower bound* (glb) of S iff q is a lower bound of S and $s \leq q$ for all $s \in S^{\downarrow}$;
7. $(Q, \leq)$ is a *lattice* iff any pair of elements in Q has a lub and a glb;

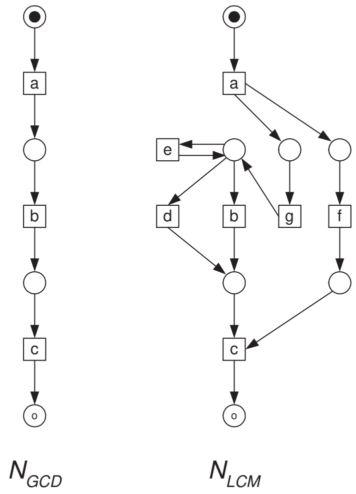

Fig. 2. The GCD and the LCM of the five OLCs shown in Figure 1.

8. $(Q, \leq)$ is a *complete* lattice iff any subset of Q has a lub and a glb.

We are not interested in distinguishing OLCs that are branching bisimilar. That is, we consider equivalence classes of OLCs under behavioral equivalence (Definition 2.11). The set of all equivalence classes is denoted $\mathcal{O}_{/\cong}$. We can lift life-cycle inheritance ($\leq_{lc}$) to $\mathcal{O}_{/\cong}$ resulting in the partial order $(\mathcal{O}_{/\cong}, \leq_{lc})$. For convenience, we refer to elements of $\mathcal{O}_{/\cong}$ as OLCs.

Definition 4.2. (MCD/GCD, MCM/LCM) Consider the inheritance partial order $(\mathcal{O}_{/\cong}, \leq_{lc})$. Let $S \subseteq \mathcal{O}_{/\cong}$ be a set of OLCs.

1. OLC N is a *Maximal Common Divisor* (MCD) of S iff (a) it is an upper bound of S (i.e., $N \in S^{\uparrow}$) and (b) for all $N' \in S^{\uparrow}$, $N' \leq_{lc} N$ implies that N' equals N (i.e., it is minimal in $S^{\uparrow}$).
2. OLC N is the *Greatest Common Divisor* (GCD) of S iff it is the lub of S.
3. OLC N is a *Minimal Common Multiple* (MCM) of S iff (a) $N \in S^{\downarrow}$ and (b) for all $N' \in S^{\downarrow}$, $N \leq_{lc} N'$ implies that N' equals N.
4. OLC N is the *Least Common Multiple* (LCM) of S iff it is the glb of S.

Note that the notions of an MCD and a GCD (an MCM and an LCM) coincide if $(\mathcal{O}_{/\cong}, \leq_{lc})$ is a complete lattice. On a first reading, Definition 4.2 might be counterintuitive: An MCD is required to be a *super*class of the OLCs in S, whereas an MCM is a *sub*class of the OLCs in S. It may be more intuitive to consider the size of the OLCs as determined by the numbers of their methods. If N_{MCD} is an MCD of two OLCs N_0 and N_1, then N_{MCD} typically contains *fewer* methods than N_0 and N_1, which conforms to the intuitive notion of an MCD. Similarly, if N_{MCM} is an MCM of N_0 and N_1, then N_{MCM} typically contains *more* methods than N_0 and N_1. Moreover, although it is straightforward to show that any MCM is a subclass under life-cycle inheritance of any MCD ($\leq_{lc}$ is transitive (Property 3.4)), an MCM is typically larger than an MCD in terms of their numbers of methods. Consider for example the OLCs of Figure 2. By Definition 4.2, N_{GCD} is an MCD of the OLCs of Figure 1 and N_{LCM} is an MCM of these OLCs. Although $N_{LCM} \leq_{lc} N_{GCD}$, N_{LCM} has more methods than N_{GCD}.

Definition 4.2 raises two interesting questions:

1. Has any set of OLCs always at least one MCD and at least one MCM?
2. Has any set of OLCs a GCD and an LCM (i.e., is $(\mathcal{O}_{/\cong}, \leq_{lc})$ a complete lattice)?

We show that the answer to the first question is (almost always) affirmative. Unfortunately, the answer to the second question is negative.

The following two properties are needed. The first one is straightforward.

Property 4.3. Let N_τ be the OLC containing one method labeled τ: $N_\tau = (\{i, o\}, \{\tau\}, \{(i, \tau), (\tau, o)\}, \{(\tau, \tau)\})$. N_τ is a superclass under life-cycle inheritance of any OLC, i.e., it is an upper bound of $\mathcal{O}_{/\cong}$ in $(\mathcal{O}_{/\cong}, \leq_{lc})$.

A set of *totally* ordered (according to $\leq_{lc}$) OLCs is called a *chain*.

Property 4.4. Let N_0 and N_1 be two OLCs in $\mathcal{O}_{/\cong}$ such that $N_0 \leq_{lc} N_1$. There is no *infinite* chain $N^0 \leq_{lc} N^1 \leq_{lc} \ldots$ of *different* OLCs $N^0, N^1, \ldots \in \mathcal{O}_{/\cong}$ such that $N_0 \leq_{lc} N^0 \leq_{lc} N^1 \leq_{lc} \ldots \leq_{lc} N_1$.

Proof. Let N and N' be two OLCs with $N \leq_{lc} N'$. The following three observations are important. First, $\alpha(N') \subseteq \alpha(N)$. Second, if N and N' are different, then $\alpha(N') \subset \alpha(N)$. Third, $\alpha(N) \setminus \alpha(N')$ is finite.

Let $N^0 \leq_{lc} N^1 \leq_{lc} \ldots$ be an infinite chain of different OLCs $N^0, N^1, \ldots$ such that $N_0 \leq_{lc} N^0 \leq_{lc} N^1 \leq_{lc} \ldots \leq_{lc} N_1$. It follows from the first two of the above observations that $\alpha(N_1) \subseteq \ldots \subset \alpha(N^1) \subset \alpha(N^0) \subseteq \alpha(N_0)$. The third observation above states that $\alpha(N_0) \setminus \alpha(N_1)$ is finite, yielding a contradiction. □

It follows immediately from the previous two properties that any non-empty set of OLCs has an MCD. The empty set does not have an MCD because $\mathcal{O}_{/\cong}$ is infinite and does not have minimal elements. Any finite set of OLCs has an MCM. First, OLC N_τ of Property 4.3 is an MCM of the empty set. Second, consider a non-empty finite set $\{N_0, N_1, \ldots, N_{n-1}\}$ of n OLCs. Let N_∂ be the OLC that is constructed from the variants as follows. The source place i of N_∂ has n output transitions, one for each variant. Each of them has a unique method label that does not occur in the alphabets of any of the variants. The source place of each variant is given a new identifier and connected as an output place to one of the n new transitions. In this way, the new transitions act as guards for the n original variants. The sink places of the n variants are simply fused together, yielding the sink place o of N_∂. Clearly, N_∂ is a subclass of each variant; by blocking all new transitions except one which is hidden, one obtains an OLC branching bisimilar to one of the variants. Based on Property 4.4, we may conclude that an MCM of the n variants exists. The above deliberations lead to the following theorem, answering the first question posed above.

Theorem 4.5. (Existence of an MCD and an MCM) Let $S \subseteq \mathcal{O}_{/\cong}$ be a set of OLCs. If S is *non-empty*, it has an MCD; if S is *finite*, it has an MCM.

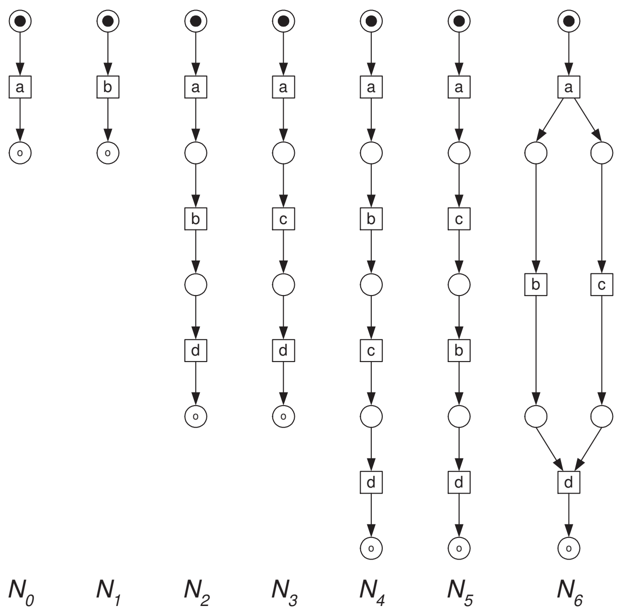

Fig. 3. Seven object life cycles.

As already mentioned, the answer to the second question posed above is negative. A set of OLCs may have two or more different MCDs, which means that it has no GCD. Similarly, a set of OLCs may have two or more different MCMs and, thus, no LCM. Consider OLCs N_4 and N_5 of Figure 3. They have at least two MCDs. It is easy to verify that both OLC N_2 and OLC N_3 are MCDs of N_4 and N_5. Each one is a superclass of both N_4 and N_5 and, in both cases, there is not a smaller (according to $\leq_{lc}$) candidate. Similarly, the two OLCs N_2 and N_3 in Figure 3 have more than one MCM. Each of the OLCs N_4, N_5, and N_6 is an MCM of N_2 and N_3. Note that hiding method c in any of the OLCs N_4, N_5, and N_6 yields an OLC equivalent to N_2. Hiding method b in any of the OLCs N_4, N_5, and N_6 yields an OLC equivalent to N_3. Clearly, in each case, there is no larger candidate.

Based on the examples in Figure 3, we conclude that a given set of OLCs can have several MCDs and MCMs. The reason that there is not a GCD for N_4 and N_5 is that they do agree on the presence of the methods b and c, whereas they do not agree on their ordering. The reason that there is not an LCM for N_2 and N_3 is that there are several ways to add methods b and c to a common subclass. However, in many situations, there is one unique MCD, which is therefore the GCD, and one unique MCM, the LCM. For example, the five variants shown in Figure 1 have a GCD and an LCM, namely the nets N_{GCD} and N_{LCM} of Figure 2, respectively. There are situations where it is quite easy to pinpoint the GCD and/or the LCM of a set of OLCs. If the set forms a *chain*, i.e., the OLCs are totally ordered according to the $\leq_{lc}$ relation, then the least element is the LCM and the greatest element is the GCD. Second, if one OLC is a superclass of all the other OLCs, then this variant is the GCD. Note that the five OLCs of Figure 1 satisfy this requirement. Third, if one OLC is a subclass of all the other variants, then this OLC is the LCM. Fourth, if two variants have no methods in common, then the GCD equals the empty OLC N_τ of Property 4.3. Finally, if the OLCs have nothing in common (i.e., with respect to internal places, transitions, and labels) and always start with a real method (i.e., a non-τ-labeled transition), then the LCM is simply the union of all OLCs (where the union means the element-wise union of the tuples defining the OLCs).

Property 4.6. Let $N_0, N_1, \ldots, N_{n-1}$, with n a positive natural number, be n OLCs.

1. If $N_0 \leq_{lc} N_1 \leq_{lc} \ldots \leq_{lc} N_{n-1}$, then N_0 is the LCM and N_{n-1} is the GCD of $N_0, \ldots, N_{n-1}$.
2. If, for all k with $0 \leq k < n$, $N_k \leq_{lc} N_0$, then N_0 is the GCD of $N_0, \ldots, N_{n-1}$.
3. If, for all k with $0 \leq k < n$, $N_0 \leq_{lc} N_k$, then N_0 is the LCM of $N_0, \ldots, N_{n-1}$.
4. If, for some j and k with $0 \leq j < k < n$, $\alpha(N_j) \cap \alpha(N_k) = \emptyset$, then N_τ of Property 4.3 is the GCD of $N_0, \ldots, N_{n-1}$.
5. If, for all j and k with $0 \leq j < k < n$, $\alpha(N_j) \cap \alpha(N_k) = \emptyset$ and $(P_j \cup T_j) \cap (P_k \cup T_k) = \{i, o\}$ and, for all k with $0 \leq k < n$ and all transitions $t \in i \overset{N_k}{\bullet}$, t has a label different from τ, then $N_\partial = \bigcup_{0 \leq k < n} N_k$ is the LCM of $N_0, \ldots, N_{n-1}$. (Note the similarity between N_∂ in this property and N_∂ as defined before Theorem 4.5.)

Proof. The first three properties follow immediately from Definitions 4.1 and 4.2.

To prove the fourth property, let N' be an arbitrary superclass of $N_0, N_1, \ldots, N_{n-1}$. Consider two variants N_j and N_k, with $0 \leq j < k < n$, such that $\alpha(N_j) \cap \alpha(N_k) = \emptyset$. Since $N_j \leq_{lc} N'$, it follows that $\alpha(N') \subseteq \alpha(N_j)$; similarly, $\alpha(N') \subseteq \alpha(N_k)$. Hence, it follows that $\alpha(N') \subseteq \alpha(N_j) \cap \alpha(N_k) = \emptyset$, which means that $\alpha(N') = \emptyset$. Consequently, N' equals N_τ, which means that N_τ is the GCD of the set of variants N_0 through N_{n-1}.

To prove the last property, we first show that N_∂ is a subclass of each of the variants. Consider a variant N_k, for some k with $0 \leq k < n$. Since for all j with $0 \leq j < n$ and $j \neq k$, $\alpha(N_j) \cap \alpha(N_k) = \emptyset$, $(P_j \cup T_j) \cap (P_k \cup T_k) = \{i, o\}$, and all transitions $t \in i \overset{N_j}{\bullet}$ have a label different from τ, blocking all transitions in $i \overset{N_\partial}{\bullet} \setminus i \overset{N_k}{\bullet}$ in N_∂, yields a marked net branching bisimilar to N_k. Hence, $N_\partial \leq_{lc} N_k$, which means that it is a subclass of all n variants. Second, we prove that any OLC N' that is a subclass of all the variants is also a subclass of N_∂. Assume that N is a subclass of all variants. Let, for all k with $0 \leq k < n$, I_k and H_k be sets of method labels such that $(\tau_{I_k} \circ \partial_{H_k}(N'), [i]) \sim_b (N_k, [i])$ (see Definition 3.3-4 (Life-cycle inheritance)). Let $I = \bigcup_{0 \leq k < n} I_k$ and $H = \bigcup_{0 \leq k < n} H_k$. Clearly, $(\tau_I \circ \partial_H(N'), [i]) \sim_b (N_\partial, [i])$, because each label in H or I appears in the alphabet of precisely one of the n variants. Hence, $N' \leq_{lc} N_\partial$. Combining the results derived so far yields that N_∂ is the LCM of the set of variants N_0 through N_{n-1}. $\square$

5 How about the other three notions of inheritance?

The results presented in Section 4 are restricted to life-cycle inheritance. In this section, we explore the other three notions of inheritance. First, we define the concepts MCD, MCM, GCD, and LCM for each of the four notions of inheritance.

Definition 5.1. (MCD$_*$, MCM$_*$, GCD$_*$, and LCM$_*$) Let S be some set of OLCs in $\mathcal{O}_{/\cong}$. OLC N is an MCD$_*$, MCM$_*$, GCD$_*$, or LCM$_*$ of S in $(\mathcal{O}_{/\cong}, \leq_*)$ with $* \in \{pt, pj, pp, lc\}$ iff the corresponding requirement stated in Definition 4.2 holds with respect to the corresponding notion of inheritance.

Note that MCD$_{lc}$, MCM$_{lc}$, GCD$_{lc}$, and LCM$_{lc}$ coincide with the concepts introduced in Section 4. As an example of this definition, consider the five variants of Figure 1. It is easy to see that N_0 is the GCD$_{pj}$ of $\{N_2, N_3, N_4\}$, the GCD$_{pt}$ of $\{N_1, N_2\}$, and the GCD$_{pp}$ of $\{N_0, N_2\}$.

The questions raised in previous section arise again: Do MCD$_*$, MCM$_*$, GCD$_*$, and LCM$_*$ exist for $* \in \{pt, pj, pp, lc\}$?

In Theorem 4.5, it has been shown that any non-empty set of variants always has an MCD$_{lc}$. Properties 4.3 and 4.4 carry over to projection inheritance. Thus, we arrive at the following theorem.

Theorem 5.2. (Existence of MCD$_{pj}$) Any *non-empty* set of OLCs $S \subseteq \mathcal{O}_{/\cong}$ has an MCD$_{pj}$ in $(\mathcal{O}_{/\cong}, \leq_{pj})$.

An MCD$_{pj}$ of the five variants of Figure 1 is the sequential OLC containing just the methods a and c. Note that N_0 is not an MCD$_{pj}$, because in N_1 it is possible to bypass b, i.e., N_{GCD} of Figure 2 is not an MCD under projection inheritance.

Unfortunately, MCD$_{pt}$, MCD$_{pp}$, MCM$_{pj}$, MCM$_{pt}$, and MCM$_{pp}$ are not guaranteed to exist. We use the variants shown in Figure 3 to give counterexamples.

Consider N_0 and N_1. There is no MCD$_{pt}$ for these two variants. Suppose that N is an MCD$_{pt}$ of the set $\{N_0, N_1\}$. N should be a superclass of both N_0 and N_1 under protocol inheritance. This implies that the alphabet of N is a subset of the intersection of the alphabets of N_0 and N_1. Since the alphabets of these two variants are disjoint, the alphabet of N is the empty set. There is just one OLC (modulo branching bisimilarity) that has the empty alphabet. This is OLC N_τ of Property 4.3. However, N_0 is not a subclass of N_τ with respect to protocol inheritance because encapsulating method a does not yield N_τ. (In fact, encapsulating a does not yield an OLC.) Therefore, there cannot be an MCD$_{pt}$ for OLCs N_0 and N_1 of Figure 3. It follows immediately from the definition of protocol/projection inheritance (Definition 3.3-3) that this example also implies that MCD$_{pp}$ does not always exist.

To prove that there may be sets of OLCs for which there is no MCM$_{pj}$, we use the variants N_4 and N_5 of Figure 3. Suppose that N is a subclass of both N_4 and N_5 under projection inheritance. The alphabet of N will include $\{a, b, c, d\}$. Let I be the set of methods in N but not in N_4 and N_5, i.e., $I = \alpha(N) \setminus \{a, b, c, d\}$. By the definition of projection inheritance, we find that $(\tau_I(N), [i]) \sim_b (N_4, [i])$ and $(\tau_I(N), [i]) \sim_b (N_5, [i])$. Hence, since $\sim_b$ is an equivalence, $N_4 \cong N_5$; that is, the two variants are equivalent modulo branching bisimilarity. Clearly, this is a contradiction. Therefore, N_4 and N_5 cannot have a common subclass under projection inheritance. As a result, they have no MCM_{pj}. It follows immediately from the definition of protocol/projection inheritance that there is also no MCM_{pp} for N_4 and N_5.

It remains to be shown that an MCM_{pt} does not necessarily exist. Consider the set S of OLCs $\{N_3, N_4\}$. Assume that N is a subclass of N_3 and N_4 under protocol inheritance. Clearly, $\alpha(N)$ is a superset of $\{a, b, c, d\}$. Let H be $\alpha(N) \setminus \{a, b, c, d\}$; that is, H contains the methods added to N_4 to obtain N, whereas $H \cup \{b\}$ contains the methods added to N_3 to obtain N. It follows from the definition of protocol inheritance that $(\partial_H(N), [i]) \sim_b (N_4, [i])$ and that $(\partial_{H\cup\{b\}}(N), [i]) \sim_b (N_3, [i])$. The definition of branching bisimilarity implies that $(N_3, [i]) \sim_b (\partial_{H\cup\{b\}}(N), [i]) \sim_b (\partial_{\{b\}}(\partial_H(N)), [i]) \sim_b (\partial_{\{b\}}(N_4), [i])$. The latter is the process that can only execute an a and then deadlocks. This is clearly not branching bisimilar to N_3. Hence, we have again a contradiction, showing that N_3 and N_4 cannot have a common subclass under protocol inheritance. This, in turn, implies that $\{N_3, N_4\}$ does not have an MCM_{pt}.

The counterexamples given show that MCD_{pt}, MCD_{pp}, MCM_{pj}, MCM_{pt}, and MCM_{pp} are not guaranteed to exist. Consequently, also GCD_{pt}, GCD_{pp}, LCM_{pj}, LCM_{pt}, and LCM_{pp} may not exist for a given set of variants. In the previous section, it has already been shown that GCD_{lc} and LCM_{lc} do not need to exist. An argument similar to the one used in the previous section shows that N_4 and N_5 in Figure 3 do not have a GCD_{pj}. Thus, also GCD_{pj} does not necessarily exist.

It remains to generalize Property 4.6 to the other notions of inheritance. The proof is omitted because it is similar to the proof of Property 4.6.

Property 5.3. Let $N_0, N_1, \ldots, N_{n-1}$, with n some positive natural number, be n OLCs and let $* \in \{pt, pj, pp, lc\}$.

1. If $N_0 \leq_* N_1 \leq_* \ldots \leq_* N_{n-1}$, then N_0 is the LCM_* and N_{n-1} is the GCD_* of $N_0, \ldots, N_{n-1}$.
2. If, for all k with $0 \leq k < n$, $N_k \leq_* N_0$, then N_0 is the GCD_* of $N_0, \ldots, N_{n-1}$.
3. If, for all k with $0 \leq k < n$, $N_0 \leq_* N_k$, then N_0 is the LCM_* of $N_0, \ldots, N_{n-1}$.
4. If, for some j and k with $0 \leq j < k < n$, $\alpha(N_j) \cap \alpha(N_k) = \emptyset$, then N_τ of Property 4.3 is the GCD_{pj} and the GCD_{lc} of $N_0, \ldots, N_{n-1}$.
5. If, for all j and k with $0 \leq j < k < n$, $\alpha(N_j) \cap \alpha(N_k) = \emptyset$ and $(P_j \cup T_j) \cap (P_k \cup T_k) = \{i, o\}$ and, for all k with $0 \leq k < n$ and all transitions $t \in i \overset{N_k}{\bullet}$, t has a label different from τ, then $N_\partial = \bigcup_{0 \leq k < n} N_k$ is the LCM_{pt} and the LCM_{lc} of $N_0, \ldots, N_{n-1}$.

6 Virtual OLCs and the Dedekind-MacNeille completion

As we have seen, each of the four inheritance relations provides a partial order on OLCs but none of these orders is a (complete) lattice. If the inheritance relations would have been complete lattices, there would have been a GCD and an LCM for any set of OLCs under any of the inheritance relations. It does not make any sense to try to modify the inheritance relations into lattices. The four relations have been carefully chosen and any attempt to transform them into lattices would reduce their applicability. If a set of OLCs has no GCD/LCM, one could settle for an MCD/MCM. However, also the MCD/MCM do not always exist, particularly for the more restrictive forms of inheritance. Fortunately, the *Dedekind-MacNeille completion* [17, 11] can be used to extend the inheritance partial orders to complete lattices. The Dedekind-MacNeille completion provides the smallest complete lattice that embeds a given partial order.

We illustrate the concepts of this section using the seven OLCs shown in Figure 4(a). Transitions without a label correspond to τ-labeled transitions (i.e., silent steps). Figure 4(b) shows the ordering relations between these OLCs under life-cycle inheritance. An OLC N is a superclass of OLC N' (i.e., $N' \leq_{lc} N$) if and only if there is a path of downward going lines from N to N'. The unconnected line segments illustrate that the seven depicted OLCs form only a part of the larger partial order $(\mathcal{O}_{/\cong}, \leq_{lc})$. Note that each element in the partial order in fact corresponds to an equivalence class of OLCs modulo branching bisimilarity.

Consider the set S of OLCs $\{N_0, N_1, N_2\}$. The elements of S are all upper bounds of the OLC sets $S_0 = \{N_3, N_4\}$ and $S_1 = \{N_3, N_4, N_5, N_6\}$; it is not difficult to see that ${S_0}^{\uparrow} = {S_1}^{\uparrow} = S$ (see Definition 4.1). S_0 and S_1 have no lub, because N_1 and N_2 are incomparable under life-cycle inheritance. In terms of Definition 4.2 (MCD/GCD, MCM/LCM), N_1 and N_2 are MCDs of S_0 and S_1, but N_0 is not; moreover, S_0 and S_1 have no GCD. Similarly, N_3, N_4, N_5, and N_6 are lower bounds and MCMs of the OLCs in S, whereas S has no glb or LCM. The reason for all this is that N_3, N_4, N_5, and N_6 agree on the presence of methods a and b but not on their ordering.

Essential in the Dedekind-MacNeille completion is the notion of *cuts*.

Definition 6.1. (Cut) Let $(Q, \leq)$ be a partial order and $A, B \subseteq Q$. (A, B) is a cut of Q if and only if $A^{\uparrow} = B$ and $A = B^{\downarrow}$.

Consider Figure 4(b). It is easy to see that $(\{N_3, N_4, N_5, N_6, \ldots\}, \{N_0, N_1, N_2\})$ is a cut, where the dots represent the subclasses of N_3, N_4, N_5, and N_6 not shown in the figure; the pair $(\{N_3, N_4, N_5, N_6, \ldots\}, \{N_0, N_1\})$ is not a cut.

We need one more definition to formalize the Dedekind-MacNeille completion.

Definition 6.2. (Order-isomorphy) Partial orders $(Q, \leq)$ and $(Q', \leq')$ are *order-isomorphic* iff there exists a bijective function $\phi : Q \rightarrow Q'$ such that, for any $x, y \in Q$, $x \leq y$ iff $\phi(x) \leq' \phi(y)$.

Theorem 6.3. (Dedekind-MacNeille completion [17, 11]) Let $(Q, \leq)$ be a partial order. Let $(Q^c, \leq^c)$ be the partial order with Q^c the set of all cuts of Q and $\leq^c$ the ordering such that, for any (A_1, B_1) and (A_2, B_2) in Q^c, $(A_1, B_1) \leq^c (A_2, B_2)$ iff $A_1 \subseteq A_2$. Order $(Q^c, \leq^c)$ is the smallest complete lattice containing an ordered subset that is order-isomorphic with $(Q, \leq)$.

An element (A, B) of Q^c corresponds to an element q of Q iff $A \cap B = \{q\}$; it has no corresponding element in Q iff $A \cap B = \emptyset$. If $S^c = \{(A, B) \in Q^c \mid A \cap B \neq \emptyset\}$, then $(S^c, \leq^c \cap (S^c \times S^c))$ is order-isomorphic with $(Q, \leq)$.

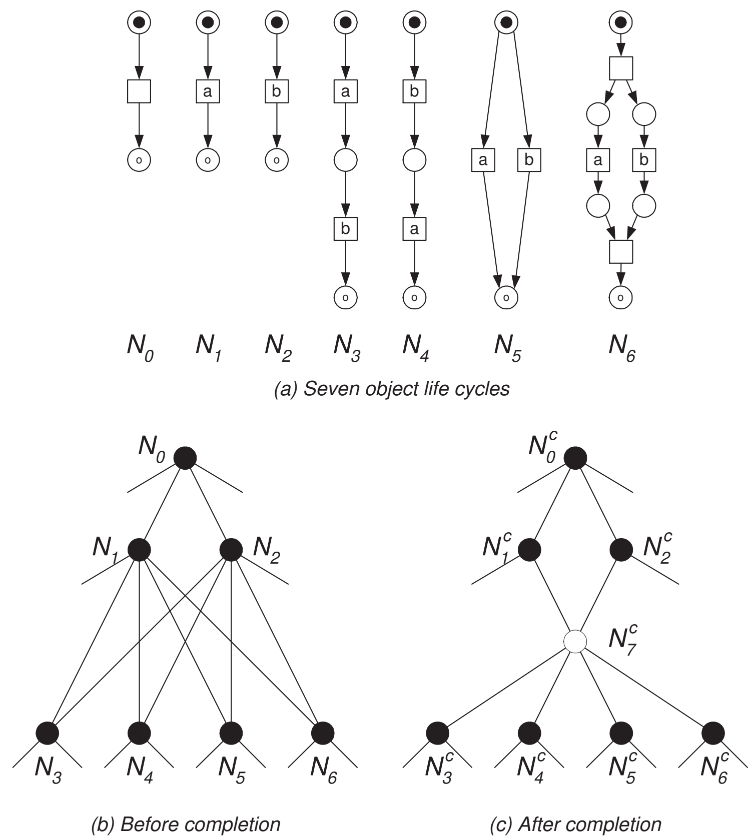

Fig. 4. Seven OLCs and their ordering under life-cycle inheritance before and after completion.

The construction of lattice $(Q^c, \leq^c)$ is known as the Dedekind-MacNeille completion. $(Q^c, \leq^c)$ is order-isomorphic with $(Q, \leq)$ if $(Q, \leq)$ is already a complete lattice. The cuts corresponding to elements of Q are called *concrete* elements; other cuts are called *virtual* elements. The Dedekind-MacNeille completion can be applied to the four inheritance partial orders.

Definition 6.4. (Dedekind-MacNeille completion) For $* \in \{pt, pj, pp, lc\}$, $(\mathcal{O}^c_*, \leq^c_*)$ is the Dedekind-MacNeille completion of partial order $(\mathcal{O}_{/\cong}, \leq_*)$.

If we apply the Dedekind-MacNeille completion to the partial order $(\mathcal{O}_{/\cong}, \leq_{lc})$, we obtain the complete lattice (partially) shown in Figure 4(c). Elements N^c_0 through N^c_7 are cuts: For example, $N^c_0 = (\{N_0, N_1, N_2, N_3, N_4, N_5, N_6, \ldots\}, \{N_0\})$, $N^c_1 = (\{N_1, N_3, N_4, N_5, N_6, \ldots\}, \{N_0, N_1\})$, $N^c_4 = (\{N_4, \ldots\}, \{N_0, N_1, N_2, N_4\})$, and $N^c_7 = (\{N_3, N_4, N_5, N_6, \ldots\}, \{N_0, N_1, N_2\})$. The black nodes in the completion of Figure 4(c) are concrete; the corresponding OLCs in Figure 4(b) can be obtained as explained in Theorem 6.3: for example, for cut N^c_1, $\{N_1, N_3, N_4, N_5, N_6, \ldots\} \cap \{N_0, N_1\}) = \{N_1\}$. Node N^c_7 is virtual; it does not correspond to an OLC: $\{N_3, N_4, N_5, N_6, \ldots\} \cap \{N_0, N_1, N_2\} = \emptyset$.

Theorem 6.5. Consider the Dedekind-MacNeille completion $(\mathcal{O}_*^c, \leq_*^c)$, with $* \in \{pt, pj, pp, lc\}$. Let $S \subseteq \mathcal{O}_{/\cong}$ be some set of OLCs; let $S^c \subseteq \mathcal{O}_*^c$ be the set of corresponding elements in $\mathcal{O}_*^c$.

1. Let N_{GCD}^c be the lub of S^c in $(\mathcal{O}_*^c, \leq_*^c)$. If N_{GCD}^c is virtual, then S has no GCD_* in $(\mathcal{O}_{/\cong}, \leq_*)$; if N_{GCD}^c is concrete, then the corresponding element N_{GCD} in $\mathcal{O}_{/\cong}$ is the GCD_* of S in $(\mathcal{O}_{/\cong}, \leq_*)$.
2. Let N_{LCM}^c be the glb of S^c in $(\mathcal{O}_*^c, \leq_*^c)$. If N_{LCM}^c is virtual, then S has no LCM_* in $(\mathcal{O}_{/\cong}, \leq_*)$; if N_{LCM}^c is concrete, then the corresponding element N_{LCM} in $\mathcal{O}_{/\cong}$ is the LCM_* of S in $(\mathcal{O}_{/\cong}, \leq_*)$.

Proof. It follows directly from Theorem 6.3 and Definitions 4.2 and 6.4. □

Theorem 6.5 illustrates that the Dedekind-MacNeille completion can be used to construct a virtual GCD_* or LCM_* for a set of OLCs if *and only if* it has no concrete $\mathrm{GCD}_*/\mathrm{LCM}_*$. A virtual $\mathrm{GCD}_*/\mathrm{LCM}_*$ cannot be drawn as an ordinary P/T-net. However, it can be expressed in terms of concrete OLCs. Consider again the virtual OLC $N_7^c = (\{N_3, N_4, N_5, N_6, \ldots\}, \{N_0, N_1, N_2\})$ of Figure 4(c). Let A and B be the first and second element of N_7^c, respectively. Sets A and B are the sets of all OLCs corresponding to the *concrete* lower bounds and *concrete* upper bounds of N_7^c in the completion, respectively. Note that the maximal elements of A and the minimal elements of B correspond to MCMs of B and MCDs of A, respectively. Virtual OLC N_7^c provides a good characterization of the GCD of N_3, N_4, N_5, and N_6, and of the LCM of N_0, N_1, and N_2. Algorithms for computing the Dedekind-MacNeille completion, see for example [9], can be used to compute (virtual) GCDs and LCMs.

7 Applications and conclusion

We have focused on the theoretical foundations for GCDs and LCMs of OLCs based on various notions of inheritance. The results are not only intriguing from a theoretical point of view. They have many applications. In component-based software development, workflow management, ERP reference models, and electronic-trade procedures, there is a constant need for identifying commonalities and differences. To conclude this paper, we discuss some of these applications.

Object-oriented methods such as *UML* [10] emphasize reuse and offer various inheritance notions. However, there is no agreement on the meaning of inheritance when considering the dynamic behavior of objects. The inheritance relations in this paper focus on dynamics [8]. One application of the GCD and LCM notions in the context of UML is the following. UML allows for the specification of sequence and collaboration diagrams. Both types of diagrams are used to describe use cases and typically describe one of many possible scenarios. A scenarios is easily translated to an OLC. The GCD of the resulting set of OLCs provides a succinct OLC capturing the behavior all scenarios agree on. The LCM of the set captures all possible behaviors generated by any of the scenarios.

Projection inheritance has been applied in the context of *component-based software architectures* [5]. One of the central issues when dealing with components is the question whether a component "fits." The framework of [5] focuses on the external behavior of a component. The question whether a component "fits" is easily expressed using inheritance. The application potential of the GCD and the LCM of a set of components is promising. The GCD can be used to deduce commonalities for a given set of similar components. The LCM can be used to construct the smallest component that can replace any of the components.

From a conceptual viewpoint, a *workflow procedure* is very similar to an OLC. Workflow management systems are driven by models that describe the life cycle of a case (e.g., insurance claim, order, or tax declaration) [1, 2, 14]. We applied the inheritance notions in the context of workflow change [4]. Using a number of construction rules, we can construct subclasses of a given workflow (i.e., correctness-by-construction). These rules allow for the automatic migration of cases from sub- to superclass and vice versa. A problem of workflow management systems supporting multiple variants of a workflow (e.g., InConcert (TIBCO) and Ensemble (Filenet)) is the lack of aggregated management information. Using the techniques of this paper, we can calculate the GCD and LCM of a set of variants. These variants may be the result of ad-hoc or evolutionary workflow changes. By migrating the status of every case residing in any of the variants to the GCD and/or LCM, one obtains aggregated management information, i.e., one diagram containing condensed information on the work in progress.

The applicability of the techniques presented in this paper is not limited to workflow within one organization. Especially *interorganizational workflows* [3] and *electronic-trade procedures* [16] can benefit from notions such as the GCD and the LCM. In [3], the notion of a view is introduced. A view is the workflow as seen by one of the parties involved. The GCD of all views is the contract all parties should agree upon. The LCM is the actual workflow being executed. The interested reader is referred to a technical report for more details [3].

Enterprise Resource Planning (ERP) systems such as SAP, Baan, Peoplesoft, and JD Edwards use *reference models* to describe and enact "best practices," i.e., proven business process models are used to drive these systems. Whenever an ERP system is installed, a considerable amount of customization is needed to adapt either the business processes inside the enterprise to the ERP system or vice versa. To determine the amount of customization, the reference model needs to be compared to the desired or actual business process. The GCD can be used to determine the commonalities between both processes and is a good predictor for the customization efforts required.

Another application of GCDs and LCMs is the *unification of procedures* in Europe. Consider for example labor mobility in Europe; a harmonization of national procedures with respect to health insurance, pensions, and so on is needed so that people can move from one EU country to another without bureaucratic confusion. Another example is the unification of financial processes resulting from the introduction of the Euro.

The applications briefly introduced in this final section show the relevance of the questions tackled in this paper. It remains for future work to study these applications in more detail.

Acknowledgment We thank the anonymous referees for their useful comments.

References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, *Lecture Notes in Computer Science* 1248, pages 407–426. Springer, Berlin, Germany, 1997.
2. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
3. W.M.P. van der Aalst. Inheritance of Interorganizational Workflows: How to Agree to Disagree Without Loosing Control? BETA Working Paper Series, WP 46, Eindhoven University of Technology, The Netherlands, 2000.
4. W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An approach to tackling problems related to change. To appear in *Theoretical Computer Science*.
5. W.M.P. van der Aalst, K.M. van Hee, and R.A. van der Toorn. Component-Based Software Architectures: A Framework Based on Inheritance of Behavior. To appear in *Science of Computer Programming*.
6. J.C.M. Baeten and W.P. Weijland. *Process Algebra. Cambridge Tracts in Theoretical Computer Science* 18. Cambridge University Press, Cambridge, UK, 1990.
7. T. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. PhD thesis, Eindhoven University of Technology, The Netherlands, 1998.
8. T. Basten and W.M.P. van der Aalst. Inheritance of Behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001.
9. K. Bertet, M. Morvan, and L. Nourine. Lazy MacNeille Completion of a Partial Order. In G. Mineau and A. Fall, editors, *Proc. of the 2nd Int. Symp. on Knowledge Retrieval, Use and Storage for Efficiency, KRUSE '97*, pages 72–81, 1997
10. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, MA, 1998.
11. B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, UK, 1990.
12. J. Desel and J. Esparza. *Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science* 40. Cambridge University Press, Cambridge, UK, 1995.
13. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.
14. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. Int. Thomson Computer Press, London, UK, 1996.
15. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley, Reading, MA, 1998.
16. R.M. Lee. Distributed Electronic Trade Scenarios: Representation, Design, Prototyping. *International Journal of Electronic Commerce*, 3(2):105–120, 1999.
17. H.M. MacNeille. Partially ordered sets. *Transactions of the American Mathematical Society*, 42:416–460, 1937.
18. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
19. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, *Lecture Notes in Computer Science* 1491. Springer, Berlin, Germany, 1998.
20. H.M.W. Verbeek and W.M.P. van der Aalst. Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, *Lecture Notes in Computer Science* 1825, pages 475–484. Springer, Berlin, Germany, 2000. http://www.tm.tue.nl/it/woflan.