BEHAVIOURAL SUBTYPING AND PROPERTY PRESERVATION FOR ACTIVE OBJECTS

Nabil Hameurlain

LIUPPA Laboratory, University of Pau, Avenue de l'Université 64012, Pau, France.

nabil.hameurlain@univ-pau.fr, <http://www.univ-pau.fr/~hameur>

Abstract: Behavioural typing and subtyping has proved to be a very useful concept for the support of incremental reuse in the area of object-oriented (O-O) languages. With the emergence of formalisms integrating the O-O approach and concurrency, yielding systems of active objects, the question arises about how behavioural subtyping may be supported by such formalisms. We present a formal framework for the definition of behavioural subtyping for active objects. This framework is based upon the preorder relations which are considered in the study of concurrent systems, allowing to define various subtyping relations. We investigate the property preservation to the degree of change of these subtyping relations, safety and liveness properties are preserved up to different extents.

Keywords: object-orientation, behavioural substitutability, subtyping, Petri nets.

1. Introduction

Inheritance is the main concept in object-oriented programming languages. It is a fundamental relation between classes, and it refers to the reuse of some (or all) elements of the definition of a class by another one. Inheritance is a matter of structure sharing and is mainly related to implementation issues. It can be applied in the incremental development of large systems: small basic system is defined initially and then additional functionality is stepwise added until designing the complete system. However it has been pointed out that inheritance entails the occurrence of many difficult problems that prohibit reuse of code containing interaction mechanisms, that is the addition of a new functionality may interfere with the old ones in an undesired way.

Subtyping relationship among the subclass and superclass has proved to be a very useful concept for the support of incremental reuse in the area of object-oriented (O-O) languages. Subtyping refers to the use of components, according to the substitution principle enunciated by P. Wegner as follows (Wegner and Zdonik, 1988): "*s is a subtype of t if an instance of type s may be substituted when an instance of type t is expected*". It is a matter of semantics and is mainly related to behaviour analysis and validation.

The emergence of formalisms integrating O-O approach with concurrency, yielding systems of active objects, gives the possibility to use theoretical tools, such as the notion of preorders, for the study of the behavioural subtyping supported by such formalisms. A number of proposals for behavioural subtyping relations have been made (see conclusion for a discussion of related work), using for instance Petri nets or labelled transition systems as semantic basis. In particular, Nierstrasz (Nierstrasz, 1993) proposes a form of substitutability called *request substitutability* preorder for active objects. Objects provide a service to their environment, and a service is determined by the service requests, which can be accepted. Thus one object is substitutable for another if it supports the same service.

This paper aims to provide behavioural subtyping relations for active objects based on action observation. We develop the approach presented in (Hameurlain and Sibertin, 1999), and propose some behavioural subtyping relations based on preorders, suited for modelling the behaviour extension in O-O concurrent systems according to the Wegner and Zdonik's substitutability principle (Wegner and Zdonik, 1988). We also investigate the characterisation of these behavioural subtyping relations to their degree of change by property preservation.

The paper is organised as follows: the second section presents the basic definitions about the notions based on action observation. In the third section we propose two requirements that any subtyping relation should satisfy in order to be quite easy to use in practice together with property preservation. The fourth section shows three subtyping relations that satisfy these requirements. These relations are based upon request substitutability preorder (Nierstrasz, 1993), and prove to be quite easy to use in practice. Due to space limitation, proofs are not reported in this paper, and may be found in the internal report (Hameurlain, 2001).

2. Basic Definitions

We start with the definitions of the relevant concepts underlying our behavioural subtyping relations. Labelled Petri Nets (PN for short) (Murata, 1989) are used to describe the behaviour of an object class, the methods that are invoked on and by object and their order of execution. Labelled PN

describe the behavioural type of an active object. The semantics of the behavioural subtyping relations and their soundness are based on the failures model of the process algebra CSP, where the communication events of CSP are identified with method invocations. We assume A to be the set of methods, that is the alphabet of observable actions, and $\{\lambda, \nu\}$ denotes two special unobservable actions. The symbol λ plays the usual role of an internal action, whose execution is under the control of the object alone; the symbol ν stands for action which is unobservable to a particular client of a server object, but is not under the control of the server alone; it may have to be executed together with another client of the server object.

Labelled Petri nets. A *Petri net* (P, T, W) consists of a finite set P of places, a finite set T of transitions where $P \cap T = \emptyset$, a weighting function $W : P \times T \cup T \times P \longrightarrow \mathbb{N}$. A *marked Petri net* is a 4-tuple $N = (P, T, W, M_N)$ where $M_N : P \longrightarrow \mathbb{N}$ is an initial marking. A transition $t \in T$ is enabled under a marking M , noted $M(t >)$, if $W(p, t) \leq M(p)$ for each place p . In this case t may occur, and its occurrence yields the follower marking M' , where $M'(p) = M(p) - W(p, t) + W(t, p)$, noted $M(t > M')$. The enabling and the occurrence of a sequence of transitions $\sigma \in T^*$ are defined inductively.

Let $M \in \mathbb{N}^P$ be a marking of N ; if $P' \subseteq P$, then $\text{Proj}(M, P') \in \mathbb{N}^{P'}$ denotes the restriction of M to P' . Let $\sigma \in T^*$ and $T' \subseteq T$, then $\text{Proj}(\sigma, T') \in T'^*$ is the substring of σ obtained by removing the occurrences of the elements of $T \setminus T'$.

We denote as LN the (marked, *labelled*) *Petri net* in which the events represent actions, which can be observable. Formally $LN = (P, T, W, M_N, l)$ consists of a marked Petri net $N = (P, T, W, M_N)$ with a labelling function $l : T \longrightarrow A \cup \{\lambda, \nu\}$. Let ε be the empty sequence of transitions, l is extended to an homomorphism $l^* : T^* \longrightarrow A^* \cup \{\lambda, \nu\}$ in the following way: $l(\varepsilon) = \lambda$ where ε is the empty string of T^* , and $l^*(\sigma.t) = l^*(\sigma)$ if $l(t) \in \{\lambda, \nu\}$, $l^*(\sigma.t) = l^*(\sigma).l(t)$ if $l(t) \notin \{\lambda, \nu\}$. In the following, we denote l^* by l , LN by (N, l) , and if $LN = (P, T, W, M_N, l)$ is a Petri net and l' is another labelling function of N , (N, l') denotes the Petri net (P, T, W, M_N, l') , that is N provided with the labelling l' . A sequence of actions $w \in A^* \cup \{\lambda\}$ is enabled under the marking M and its occurrence yields a marking M' , noted $M(w >> M')$, iff either $M = M'$ and $w = \lambda$ or there exists some sequence $\sigma \in T^*$ such that $l(\sigma) = w$ and $M(\sigma > M')$. (The first condition accounts for the fact that λ is the label image of the empty sequence of transitions). A marking is stable if no unobservable action λ is enabled: M stable if not $(M(\lambda >>))$. For a net N , $\text{Reach}(N) = \{M; \exists \sigma \in T^*; M_N(\sigma > M)\}$ is the set of reachable markings of N .

Definition 2.1 (Trace and language)

Let $N = (P, T, W, M_N, l)$ be a net. Then $Tr(N) = \{\sigma \in T^*; M_N(\sigma >)\}$ is the trace of N , i.e. the set of enabled transition sequences of N . The label image of the trace of N is its language $L(N) = l(Tr(N)) = \{w \in A^*; \exists \sigma \in Tr(N), l(\sigma) = w\}$.

Definition 2.2 (Failure)

Let $N = (P, T, W, M_N, l)$ be a net. Then the *failure* of the net N on T' is $F(N, T') = \{(\sigma, S); \sigma \in T^*, S \subseteq T', \text{ and there exists some marking } M \text{ such that } M_N(\sigma > M, \text{ and } \forall t \in S, \text{ not } (M(t >))\}$.

The label image of the failure of N is $F(N, l) = l(F(N, T)) = \{(w, X); w \in A^* \cup \{\lambda\}, X \subseteq A, \text{ and there exists some marking } M \text{ such that } M_N(\sigma > M, l(\sigma) = w, M \text{ stable, and } \forall a \in X, \text{ not } (M(a >))\}$.

Operations on the nets. To define our behavioural subtyping relations we need four basic operations on the nets: encapsulation, abstraction, cancellation, and parallel composition, used for testing characterisations of type substitutability:

- The *encapsulation operator* τ on $I \subseteq A$ removes some transitions from a Labelled net. Formally, given a Labelled Petri net $N = (P, T, W, M_N, l)$, for each $H \subseteq A$, $\tau_H(N) = N' = (P, T', W', M_N, l')$ such that $T' = \{t \in T, l(t) \notin H\}$, $W' = W \cap ((P \times T') \cup (T' \times P))$, and $l' = l \cap (T' \times A)$.
- The *abstraction operator* λ labels as not observable and internal actions, some transitions of a Labelled net. It introduces new non-stable states, from which the refusal sets are not taken into account for the failure semantics. Formally, given a Labelled Petri net $N = (P, T, W, M_N, l)$, for each $H \subseteq A$, $\lambda_H(N) = N' = (P, T, W, M_N, l')$ such that $l'(t) = l(t) = s$, if $t \in T$ and $s \in A \setminus H$, $l'(t) = \lambda$ else.
- The *cancellation operator* δ labels as not observable, but not internal actions, some transitions of a Labelled net. Cancellation is another kind of abstraction, which does look at the new non-stable states when computing failures. It renames transitions into v transitions. Formally, given a labelled Petri net $N = (P, T, W, M_N, l)$, for each $H \subseteq A$, $\delta_H(N) = N' = (P, T, W, M_N, l')$ such that $l'(t) = l(t) = s$, if $t \in T$ and $s \in A \setminus H$, $l'(t) = v$ else.
- The *parallel composition operator* $C_p : \text{Net} \times \text{Net} \longrightarrow \text{Net}$ computes the set of parallel compositions of traces, interleaving the non-synchronised actions and sharing the synchronised actions. Given two nets $N_1 = (P_1, T_1, W_1, M_1, l_1)$, $N_2 = (P_2, T_2, W_2, M_2, l_2)$ and $I \subseteq A_1 \cap A_2$, the parallel composition of N_1 and N_2 is denoted by a net $CP = C_{p_1}(N_1, N_2)$. It is the disjoint union of N_1 and N_2 , except for transitions with labels from I : for each $a \in I$, we combine each transition from N_1 labelled by a with each transition from N_2 labelled a .

In the rest of this paper, we use the letter C for *Client* net used for testing characterisations and S_i for *Server* nets to be substituted. A Client net C and a Server net S may be *composed* into a CP net, $CP = Cp_I(C, S)$ on $I \subseteq A_C \cap A_S$. We will always assume that Petri nets which are composed are composable without recalling the hypothesis $A_C \cap A_S \neq \emptyset$, and therefore, $Cp_I(C, S)$ is denoted simply by $Cp(C, S)$. The parallel composition operator allows to analyse the behaviour of a system in an incremental way (Vogler 1992): the behaviour of a compound net CP may be deduced from the behaviour of its components.

3. Requirements for Behavioural Subtyping

Behavioural subtyping relation is a preorder (reflexive and transitive) relation. Reflexivity allows that each object is substitutable by itself, and transitivity allows that if an object S_2 extends the capabilities of an object S_1 and an object S_3 extends the capabilities of S_2 then S_3 extends the capabilities of S_1 . Our proposal is to base subtyping relations on the preorder which have been introduced to compare the behaviour of concurrent systems, such as the language, failure, readiness, deadlock (see (Pomello, 1992), (Van Glabbeek and Goltz, 1989), (Vogler 1992) for a comparative study of these relations).

3.1 Requirements

A behavioural subtyping relation $\leq_H$ is a preorder satisfying the following requirements, where S_1, S_2 , and C are any Petri nets:

- (I) $S_2 \leq_H S_1 \Rightarrow \forall C, Cp(C, S_2) \leq_H Cp(C, S_1)$
 - (II) $S_2 \leq_H S_1 \Rightarrow \forall C (\exists \mathcal{R}_1, S_1 \mathcal{R}_1 C \Rightarrow \exists \mathcal{R}_2, S_2 \mathcal{R}_2 C)$
- Where $\mathcal{R}_i$ are behavioural preorder relations.

Notice that S_2 subtype of S_1 is denoted $S_2 \leq_H S_1$, in accordance with the idea that S_2 extends the capabilities of S_1 .

Requirement (I) asserts that $\leq_H$ is compositional for the net composition operation. This property is necessary to ensure the compatibility of the subtyping relation with the incremental design of systems; if it does not hold, there is no way to relate the behaviour of $Cp(Cp(C, S_1), S)$ and $Cp(Cp(C, S_2), S)$ when $S_2 \leq_H S_1$.

The *requirement (II)* for the behavioural subtyping formalises the principle of type substitutability by a testing scenario: the client C uses the services of the server nets S_1 and S_2 ; a server S_2 is type substitutable S_1 if every request for a service that S_1 fulfils is also satisfied by S_2 ; the consequence " $S_2 \mathcal{R}_2 C$ " formalises the substitutability principle as the impossibility for C to distinguish between the behaviours of S_1 and S_2 with

regard to the properties which are accounted for by the $\mathcal{R}_2$ relation. Indeed, the set of properties of S_1 which can be preserved by S_2 depends on the way that $\leq_H$ is defined; for instance, if $\leq_H$ is defined by an interleaving semantics, it is not reasonable to require that S_2 satisfies the same partial order properties as S_1 . Requirement (II) asserts that S_2 may be substituted for S_1 if $S_2 \leq_H S_1$ and in addition the net C which is considered satisfies $S_1 \mathcal{R}_1 C$. Indeed it is of interest to know that $S_2 \mathcal{R}_2 C$ only if S_1 behaves properly; in the opposite case, nothing is gained by knowing that S_1 and S_2 are both a wrong system!

Thus, to prove that a preorder relation is a subtyping relation for a composition operation C_p , we just have to prove that (I) it is compositional for C_p and that (II) holds for some couples of preorder relations $\mathcal{R}_1$ and $\mathcal{R}_2$. Then, the practical use of a subtyping relation is quite easy: once $S_2 \leq_H S_1$ has been established, it is enough to check whether $S_1 \mathcal{R}_1 C$ holds to be allowed to substitute S_2 for S_1 in a safe way, whatever is the net C under consideration.

The above requirements for the behavioural subtyping have a wide range of applicability and suffer no ambiguity. They provide the concept of subtype with a formal definition, and are based on a theory (the theory of the behavioural preorder relations), which is relevant for any concurrent language supporting a composition mechanism. They extend Liskov's approach (Liskov and Wing, 1994), since requirement (II) automatically determines what are the properties of S_1 which are preserved by S_2 when S_2 is a subtype of S_1 , so that no specific verification has to be done.

These two requirements do not define just one subtyping relation but allow several such relations. Faced with a given system, the designer may use the subtyping relation preserving properties, which are of importance for that system. We can divide the subtyping relations into two classes (Wehrheim, 2000): those suitable for *single access server* (a single client); or those for *shared mutable Server* (more than one client using a server); in the second class, we consider that the environment may interfere with the requested services of the client. We will present in the section 4, the best subtyping relation for concurrent object, called *optimal subtyping*: the environment cannot disturb a service for a client on a subtype-server that was possible in the super-server, and the safety and liveness properties of the client are preserved when the super-server is substituted by the subtype-server.

3.2 Property Preservation

Among the numerous preorders, which may be used to study behavioural relations between a server net and its Client, only the Language and Failure ones will be dealt with. Indeed, trace semantics are the coarsest one; they are

the less demanding and it is meaningless to compare nets whose languages are not comparable. However the trace semantics is related to the safe properties and it is often regarded as too weak for many applications because it is not able to express deadlock behaviour. Failure semantics have to be considered which are sensitive to liveness properties. First, let us define safety properties (quasi-liveness of actions, reachability of markings) and the liveness properties (deadlock-free, and liveness of the net), and then, based on trace and failure semantics, define behavioural preorder relationships between the Client and the server net related to these properties: *Compatibility* and *Transparency*. The compatibility is related to the safety property, whereas the transparency is related to the liveness property.

Definition 3.1 (safety and liveness property)

Let $N = (P, T, W, M_N, l)$ be a net.

- i) An action $a \in A$ is *quasi-live* iff $\exists t \in T, l(t) = a, \exists M \in \text{Reach}(N), M(t) >$.
- ii) N is *deadlock free* iff $\forall M \in \text{Reach}(N), \exists t \in T, M(t) >$.
- iii) A transition $t \in T$ is *live* iff $\forall M \in \text{Reach}(N), \exists M' \in \text{Reach}(N), \exists \sigma \in T^*, M(\sigma > M' \text{ and } M'(t) >$. Then, we say that a net N is live (resp. deadlock free) if and only if all its transitions are live (resp. for each reachable marking of the net N there exists an enabled transition under this marking).

Definition 3.2 (compatibility)

- i) S is *weakly compatible* with C , noted $S \leq_{wc} C$, iff $L(C) \cap A_C^* \subseteq L(CP) \cap A_C^*$;
- ii) S is *strongly compatible* with C , noted $S \leq_{sc} C$, iff $L(C) \subseteq l_C(\text{Tr}(CP))$.

The weak compatibility ensures that the enabling of the client component's actions are preserved in the compound CP , whereas the strong one ensures in addition the preservation of their sequences.

The compatibility of the server S with a client C is related to the preservation of the safety properties of the Client C in CP , which possibly accepts more services than those of C .

Property 3.1

- i) If S is *weakly compatible* with C then,
if $a \in A_C$ is quasi-live in C then a is quasi-live in CP .
- ii) If S is *strongly compatible* with C then,
if $M \in \text{Reach}(C)$ then there exists a reachable marking $M' \in \text{Reach}(CP)$ such that $M = \text{Proj}(M', P_C)$.

Similarly, relationships between the client and the server nets, related to the liveness properties, can be deduced from the transparency of the server for a client:

Definition 3.3 (transparency)

Let C be a Client, S be a Server, and let $CP = Cp(C, S)$ and $l_C(t) = s$, if $t \in T_C$ and $s \in A_C$, $l_C(t) = \lambda$ else.

- i) S is *weakly transparent* for C , noted $S \leq_{WT} C$, iff $F(CP, l_C) \subseteq F(C, l_C)$;
- ii) S is *strongly transparent* for C , noted $S \leq_{ST} C$, iff $(\sigma, S) \in F(CP, T_C) \Rightarrow (\sigma', S) \in F(C), \sigma' = \text{Proj}(\sigma, T_C)$.

The refusal sets in weak transparency guarantees that the same set of *actions* on A_C are blocked, whereas in the strong transparency, it guarantees that the same set of *transitions* of T_C are blocked. For instance it is easy to prove, if a server S is weakly [resp. strongly] transparent for a client C , then deadlock-free [resp. liveness of the services] of the net C is preserved in the compound CP .

Property 3.2

- i) If S is *weakly transparent* for C then,
if C is deadlock free then C is deadlock-free in CP .
- ii) If S is *strongly transparent* for C then $\forall t \in T_C$, if t is live in C then t is live in CP .

Strong compatibility [resp. transparency] always implies weak compatibility [resp. transparency].

Proposition 3.3

$$\leq_{SC} \Rightarrow \leq_{WC} \quad \text{and} \quad \leq_{ST} \Rightarrow \leq_{WT}$$

4. Behavioural Subtyping Relations

We are now finally ready to define behavioural subtyping relations. Our main interest is in capturing substitutability, namely the ability to replace an object with another object without losing behaviours. We are interested in objects having deterministic behaviour, since for non-deterministic behaviour checking a preorder becomes semi-decidable.

4.1 Subtyping Relations

For the definitions of our subtyping relations, we use *extension* (Brinksma et al., 1987) which is the most popular one as a choice for subtyping relation in Object-Oriented formalisms. It allows extension of functionality, that is the subtype may have new services in addition to the ones of the supertype. Extension has been used by Nierstrasz (Nierstrasz, 1993) in order to define

the Request Substitutability (RS) preorder for active objects. This relation requires that the sequences of service that an object can accept to process constitute a regular language, since when the behaviour is non-deterministic, reflexivity is lost (an object is not substitutable for itself). In our context we will use the RS preorder according to the Wegner and Zdonik's substitutability principle (Wegner and Zdonik, 1988). Then, there are basically three possibilities to treat the new services: we restrict them (encapsulation), we hide them (abstraction) or we cancel them (cancellation). First, we start to define RS preorder and then, using the abstraction operator, we define the strong subtyping relation; we prove that strong subtyping preorder is equivalent to RS preorder. The choice of the name is due to the fact that the requirement II holds for the compatibility and transparency preorders.

Definition 4.1 (Request Substitutability (RS) preorder)

Let N_1 and N_2 be two nets such that $A_1 \subseteq A_2$.

N_2 is less or equal to N_1 w.r.t *Request Substitutability (RS) preorder* denoted $N_2 \leq_{RS} N_1$, iff $L(N_1) \subseteq L(N_2)$ and $\forall (w, X') \in F(N_2, l_2)$, $w \in L(N_1)$, there exists $(w, X) \in F(N_1, l_1)$ such that $X' \subseteq X$.

If N_2 is less or equal to N_1 w.r.t *Request Substitutability*, then the language of the net N_2 must include the language of the net N_1 , and if the language of the net N_1 is performed by the net N_2 , then its possible failure must be a subset of the corresponding failures of N_1 .

Definition 4.2 (Strong Subtyping)

Let N_1 and N_2 be two nets such that $A_1 \subseteq A_2$.

N_2 is less or equal to N_1 w.r.t *Strong Substitutability*, denoted $N_2 \leq_{SS} N_1$, iff there exists a set $H \subseteq A_2 \setminus A_1$ such that $\tau_H(N_2) \leq_{RS} N_1$.

If N_2 is less or equal to N_1 w.r.t *Strong Substitutability*, then the server N_1 can be substituted by a server N_2 and the client will not be able to notice the difference since the new services added in the server N_2 are considered unobservable, through the encapsulation operator τ_H .

The following property shows that the Strong Substitutability is equivalent to the Request Substitutability.

Property 4.1

Let N_1 and N_2 be two nets such that $A_1 \subseteq A_2$. Let $H = A_2 \setminus A_1$.

$N_2 \leq_{RS} N_1$, if and only if $\tau_H(N_2) \leq_{RS} N_1$ that is $N_2 \leq_{SS} N_1$.

Example 1: As an example, consider the Bounded Buffer Server net BB shown in Figure 1 (a) and unbounded buffer Server net UB shown in Figure

1 (b). It is easy to prove that $UB \leq_{ss} BB$ holds, whereas the Bounded Buffer Server net BB shown in figure 2 (a) and the Copy-Bounded Buffer net CBB in figure 2 (b) are not related by the strong subtyping relation, i.e. $CBB \leq_{ss} BB$ does not hold. This is captured by the second type of Substitutability, called *Weak Substitutability*, which is less restrictive than Strong Substitutability.

(a) The Bounded Buffer Server net BB, $A = \{\text{push}, \text{pop}\}$;

(b) The Unbounded Buffer Server net UB, $A = \{\text{push}, \text{pop}, \text{copy}, \text{destroy}\}$.

Figure 1: UB subtype of BB.

(a) The Bounded Buffer Server net BB, $A = \{\text{push}, \text{pop}\}$;

(b) The Copy-bounded Buffer Server net CBB, $A = \{\text{copy}, \text{push}, \text{pop}\}$.

Figure 2: CBB weak subtype of BB.

Definition 4.3 (Weak Subtyping)

Let N_1 and N_2 be two nets such that $A_1 \subseteq A_2$.

N_2 is less or equal to N_1 w.r.t *Weak Substitutability*, denoted $N_2 \leq_{ws} N_1$, iff there exists a set $H \subseteq A_2 \setminus A_1$ such that $\lambda_H(N_2) \leq_{rs} N_1$.

If N_2 is less or equal to N_1 w.r.t *Weak Substitutability*, then the server N_1 can be substituted by a server N_2 and the client will not be able to notice the difference since the new services added in the server N_2 are considered unobservable, through the abstraction operator λ_H . In the example of figure 2, $CBB \leq_{ws} BB$ holds; in fact with $H = \{\text{copy}\}$, we have $\lambda_H(CBB) \leq_{rs} BB$.

Example 2: consider the Bounded Buffer Server net (BB, l_{BB}) shown in Figure 3 (a) and Copy-Destroy Buffer Server net (CDB, l_{CD}) shown in Figure 3 (b). It is easy to prove that $CDB \leq_{ss} BB$ holds. In addition, $CDB \leq_{ws} BB$ also holds. Thus a client using the services $\{\text{push}, \text{pop}\}$ will not be able to notice the difference between BB and CDB .

(a) The Bounded Buffer Server net BB , $A = \{\text{push}, \text{pop}\}$;
 (b) The Copy-Destroy Buffer Server net CDB , $A = \{\text{push}, \text{pop}, \text{copy}, \text{destroy}\}$.

Figure 3: CDB strong subtype BB .

The above example shows why we are still not at the end of defining relations. The net CDB is a server which accepts services $\{\text{push}, \text{pop}, \text{copy}, \text{destroy}\}$. The new services to be added are $\{\text{copy}, \text{destroy}\}$. Thus, as soon as another client is additionally using these new services, the client of the old services $\{\text{pop}, \text{push}\}$, might indeed notice the differences. The extension is however both weak and strong, and is adapted for a single access server. Instead, for shared servers, the third type of Substitutability, *Optimal Substitutability*, which is more restrictive than Strong Substitutability, is necessary to capture the desired substitutability.

Definition 4.4 (Optimal Subtyping)

Let N_1 and N_2 be two nets such that $A_1 \subseteq A_2$.

N_2 is less or equal to N_1 w.r.t *Optimal Substitutability*, denoted $N_2 \leq_{os} N_1$ iff $\delta_H(N_2) \leq_{rs} N_1$ for $H = A_2 \setminus A_1$.

If N_2 is less or equal to N_1 w.r.t *Optimal Substitutability*, then the server N_1 can be substituted by a server N_2 and the client will not be able to notice the difference even there exists another client using the new services added in the server N_2 . In fact, these new services are considered unobservable, through the cancellation operator δ_H and can be executed concurrently with the old one.

Example 3: First, let us consider again the example given in figure 3. $CDB \leq_{OS} BB$ does not holds. Indeed, let $H = \{\text{copy}, \text{destroy}\}$; we have $(w, \text{push}) \in F(\delta_H(CDB, l_{CD}))$ where $w = l_{CD}(t1t2t3) = \text{push.pop}$, $w \in L(BB)$ and $(w, \text{push}) \notin F(BB, l_{BB})$. A correct extension of the Bounded Buffer is shown in figure 4. The new services $\{\text{copy}, \text{destroy}\}$ can be concurrently executed with the old one. For a client which requests the push-pop services, feature the same service as before is now possible.

(a) The Bounded Buffer Server BB, $A = \{\text{push}, \text{pop}\}$

(b) The Correct Copy-Destroy Buffer Server CCDB, $A = \{\text{push}, \text{pop}, \text{copy}, \text{destroy}\}$.

Figure 4: A correct extension of BB by Optimal subtyping : $CCDB \leq_{OS} BB$.

4.2 Characterisations of Subtyping Relations

We now have three subtyping relations for extension of functionality: two for servers with single access, and one for mutable shared servers; we will see that the first one guarantees the requirement II for the compatibility and only for the weak transparency, whereas the second one guarantees the requirement II for the (weak and strong) compatibility and transparency, and the third subtyping relation guarantees this property also for shared mutable servers. Optimal subtyping is the strongest of the three behavioural subtyping relations.

Property 4.2 (hierarchy of subtyping relations)

The subtyping relations form a hierarchy: $\leq_{OS} \Rightarrow \leq_{SS} \Rightarrow \leq_{ws}$.

In the following we demonstrate that the requirements (I) and (II) hold for these preorders. First, we present a technical lemma:

Lemma 4.3

Let C be a Client net and S_1, S_2 be two Server nets such that $A_1 \subseteq A_2$. Then we have: if $S_2 \leq_{WS} S_1$ then $\text{Proj}(\text{Tr}(\text{Cp}(C, S_1)), T_C) \subseteq \text{Proj}(\text{Tr}(\text{Cp}(C, S_2)), T_C)$.

As expected, the $\leq_H$ preorders where $H \in \{\text{OS}, \text{SS}, \text{WS}\}$ are compositional for the composition operator Cp and satisfy requirement (I); for instance, extending the Supply of a Server net also extends the Supply of its composition with any Client net.

Theorem 4.5 (Requirement I)

Let S_1, S_2 be two nets such that $S_2 \leq_H S_1$ where $H \in \{\text{OS}, \text{SS}, \text{WS}\}$. Then for any net C , we have $\text{Cp}(C, S_2) \leq_H \text{Cp}(C, S_1)$.

According to the Theorem 4.5 and lemma 4.3, the $\leq_H$ preorders, $H \in \{\text{OS}, \text{SS}, \text{WS}\}$, are subtyping relations suited for modelling extension, and we have the following corollary:

Corollary 4.6 (Requirement II)

Let C be a client net, S_1, S_2 two server nets.

- i) $S_2 \leq_{WS} S_1$ and $S_1 \leq_{WC} C \Rightarrow S_2 \leq_{WC} C$.
- ii) $S_2 \leq_{WS} S_1$ and $S_1 \leq_{SC} C \Rightarrow S_2 \leq_{SC} C$.
- iii) $S_2 \leq_{WS} S_1$ and $S_1 \leq_{WT} C \Rightarrow S_2 \leq_{WT} C$.
- iv) $S_2 \leq_{SS} S_1$ and $S_1 \leq_{ST} C \Rightarrow S_2 \leq_{ST} C$.

We have proposed three relations which are suitable as behavioural subtyping relations for active objects: weak subtyping is the right relation if we would like to preserve safety properties, and strong subtyping is useful to preserve safety and liveness properties; these two subtyping relations are only suitable for servers with single access. The third subtyping relation is the strongest and guarantees the preservation of safety and liveness properties also for shared mutable servers. The results can be applied in an incremental design with inheritance by checking what kind of subtype a subclass is and then deduce what properties of the superclass are preserved. The table in the figure 5 summarises the results of the requirement II for the subtyping relations given in this paper.

Subtyping	Compatibility		Transparency		Server Access	
	Weak	Strong	Weak	Strong	Single	Shared
<i>Optimal</i>	√	√	√	√	√	√
<i>Strong</i>	√	√	√	√	√	
<i>Weak</i>	√	√	√		√	

Figure 5: Characterisations of subtyping relations.

Last but not least, many problems about the composition and the behavioural subtyping of Petri nets are decidable when the language of the nets is regular even if they are unbounded. This property is noteworthy, since most of equivalence preorder (equivalence) relations are undecidable for labelled Petri nets (Jancar, 1995). In (Hameurlain, 1998) we have shown that it is decidable whether: a language of a Petri net is regular, a server net is (strongly and/or weakly) compatible (resp. transparent for) with a client net. All these properties are proved, using construction due to Vogler (Vogler 1992), since they are equivalent to the liveness property of certain transition in the constructed net, which is decidable in Petri nets (Mayr, 1982).

5. Discussion

This paper proposes a formal framework for the definition of behavioural subtyping relations for active objects. This framework consists of requirements that any subtyping relation ought to satisfy, and it is based upon the preorder relations commonly considered in the study of concurrent systems. It is in accordance with the principle of substitutability and extends Liskov's approach, since the type of preserved properties is characterised by preorder relations (compatibility and transparency), and then it can be preserved automatically by subtyping without specific verification.

The approach presented in this paper leads to define subtyping relations for active objects having good properties; we believe that it could be fruitfully applied to Concurrent O-O languages supporting active objects where the behaviour of a compound Object may be deduced from the behaviour of its components. Especially CoOperative Objects formalism (Sibertin, 2001) views a system as a collection of active objects where the communication between objects is achieved in an asynchronous way by channel places according to the Client/Server Protocol. The results of this paper have been generalised in the context of Client/Server Petri nets (Sibertin, 1993), and the implementation of these subtyping relations should be envisaged for CoOperative Objects formalism. This work is under investigation, and constitutes the part of our future work.

Related work. The starting point of the presented approach is the process algebra correctness relation, extension (Brinksma et al., 1987), most often discussed as a subtyping relation for object-oriented systems. (Van der Aalst and Basten, 1997) have provided net transformation rules which preserve the behaviour; this approach combines the branching bisimulation equivalence and the encapsulation and abstraction operators. (Balzarotti et al., 1999) have shown that the behavioural inheritance proposed by Van der Aalst is a too strong relation, and cannot be applied in many cases. Thus (Balzarotti et al., 1999) have proposed an approach to combine the renaming, abstraction operators and the Nierstrasz' request substitutability preorder; the authors have shown that obtained subtyping relation is appropriate to solve some critical examples in the literature; unlike, no testing characterisations (composition of nets) are given, and no preservation property are investigated. Recently (Wehrheim, 2000) have investigated this topic and she proposes three behavioural subtyping taxonomically ordered in a hierarchy; one of them is suitable for shared mutable objects; both safety and liveness properties from CSP point of view are considered. This approach investigates up to which extent properties of a superclass are preserved under behavioural subtyping, and gives the characterisation of these subtyping relations by the properties they preserve from supertypes. In contrast, our behavioural subtyping relations are suited for the behaviour extension, taking into account the composition mechanism, and determine automatically the properties, which are preserved.

6. References

- Van der Aalst, W.M.P. and Basten, T. (1997). Life-cycle Inheritance: A Petri-Net-Based Approach; in *Proc. 18th International Conference on Applications and Theory of Petri Nets, ICATPN'97*, P. Azéma and G. Balbo Eds., Toulouse (F), June 1997, LNCS 1248, Springer-Verlag.
- Balzarotti, C., De Cindio, F. and Pomello, L. (1999). Observation Equivalence for the Semantics of Inheritance; In *Proc. FMOODS'99*. P. Ciancarini, A. Fantechi & R. Gorrieri eds, pp 67-82.
- Brinksma, E., Scollo, G. and Steenbergen, Ch. (1987). LOTOS specifications, their implementations and their tests. In B. Sarikaya and G.v. Bochmann, editors, *Protocol Specification, Testing and Verification VI*, pages 349-358. Elsevier.
- Van Glabbeek, R. and Goltz, U. (1989). Equivalence notions for concurrent systems and refinement of actions; *MFCS 89*, LNCS 379, Springer-Verlag.
- Hameurlain, N. (1998). Behavioural Inheritance in Petri Nets; Ph.D Thesis, Toulouse I University (F).
- Hameurlain, N. and Sibertin-Blanc, C. (1999). Behavioural Types in CoOperative Objects; *Second International Workshop on Semantics of Objects As Processes, SOAP'99*. Lisbon, Portugal. BRICS NS-99-2, Hüttel et al. (eds.), pp 29-38.

- Hameurlain, N. (2001). Behavioural Subtyping and property preservation for Object-Oriented Formalisms. Internal report, University of Pau, available at <http://www.univ-pau.fr/~hameur/report.html> as report3.ps
- Jancar, P. (1995). Undecidability of Bisimilarity for Petri Nets and some related problems. *Theoretical Computer Science*, N° 148.
- Liskov, B. H. and Wing, J. M. (1994). A Behavioral Notion of Subtyping; in *ACM Trans. on Programming Languages and Systems*, Vol 16, n° 6.
- Murata, T. (1989). Petri Nets: Properties, Analysis and Applications; *Proc. of the IEEE*, vol. 77, no. 4, pp. 541-580.
- Mayr, E. W. (1982). An algorithm for the general Petri Net Reachability Problem. *Proc. of the 13th ACM Symposium on theory of computing*.
- Nierstrasz, O. (1993). Regular types for Active Objects; in ACM sigplan Notices, 28 (10); *Proceedings of the 8th Annual Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA '93*, Washington DC, pp. 1-15.
- Pomello, L., Rozenberg, G. and Simone, C. (1992). A Survey of Equivalence Notions for Net Based System. *Advances in Petri Nets 1992*; G. Rozenberg Ed., LNCS 609, Springer-Verlag.
- Sibertin-Blanc, C. (1993). A Client-Server Protocol for the Composition of Petri Nets; *ICATPN'93*, Chicago (Il), LNCS 691, Springer-Verlag.
- Sibertin-Blanc, C. (2001). CoOperative Objects: Principles, Use and Implementation. In *Object Oriented Programming and Models of Concurrency*, G. Agha & F. De Cindio Eds, LNCS Vol 2001, Springer-Verlag.
- Vogler, W. (1992). *Modular Construction and Partial Order Semantics of Petri Nets*; LNCS 625, Springer-Verlag.
- Wehrheim, H. (2000). Behavioural Subtyping and Property preservation. In *IFIP Proc. of FMOODS'00*, Standford, USA, september 6-8.
- Wegner, P. and Zdonik, S. (1988). Inheritance as an Incremental Modification Mechanism, or What Is and Isn't Like; in *Proc. ECOOP 88*, Oslo (Norway), LNCS 322, Springer-Verlag, pp. 55-77.