Nautilus: A Concurrent Anticipatory Programming Language

P. Blauth Menezes[*], Simone A. Costa[¶], Júlio P. Machado[*] and Jaime Ramos[†]

[*]*Instituto de Informática, UFRGS, Av. Bento Gonçalves 9500, Campus do Vale, Bloco IV, CEP 91501-970, Caixa Postal: 15064, Porto Alegre, Brazil {blauth, jhapm}@inf.ufrgs.br*

[¶]*Centro de Ciências Exatas e Tecnológicas, UNISINOS, Av. Unisinos 950, CEP 93022-000, São Leopoldo, Brazil sac@exatas.unisinos.br*

[†]*CLC, Departamento de Matemática, IST, Av. Rovisco Pais, 1049-001, Lisboa, Portugal jabr@math.ist.utl.pt*

Abstract. Nautilus is a concurrent anticipatory programming language based on the object-oriented language GNOME which is a simplified and revised version of OBLOG. A semantics for Nautilus is given by Nonsequencial Automata, that is a categorial semantic domain based on labeled transition system with full concurrency, where a class of morphisms stands for anticipation. The semantics of an object in Nautilus is given by an anticipation morphism, which is viewed as a special automaton morphism where target automata, called base, is determined by the computations of a freely generated automata able to simulate any object specified over the involved attributes, and the source automata is a relabelled restriction of the base. In order to introduce the anticipation of Nautilus, some examples are presented depicting the features of the language.

Keywords: anticipation, nonsequential automata, programming language, category theory.

1 INTRODUCTION

The main purpose of this paper is to present Nautilus as a concurrent anticipatory programming language, i.e., its semantic is an anticipatory system that involves "its future", inspired by [5] and based on [8, 9]. An anticipatory system [5] is a system for which the present behavior is based on past and/or present events but also on future events built from these past, present and future events. Nautilus [8, 2, 3] is a general purpose concurrent object-based language, originally based on the language Gnome [10, 11, 14], and introduces some special features inspired by the semantic domain such as anticipation. A semantics for Nautilus is given by Nonsequential Automata [8, 7, 9, 6], which constitute a categorial [1] semantic domain based on labeled transition system with full concurrency, where a class of morphisms stands for anticipation. It is a model which satisfies the diagonal compositionality requirement, i.e., anticipations compose and distribute over system combinators.

In Nautilus, an object can be specified either as a simple object or the resulting object of an encapsulation, aggregation, anticipation or parallel composition. An action of an object in Nautilus may be a sequential or concurrent composition of clauses, executed in an atomic way. The semantics of an object in Nautilus is given by an anticipation morphism where the target automata, called base, is determined by the computations of a freely generated automata able to simulate any object specified over the involved attributes, and the source automata is a relabelled restriction of the base. An anticipation maps transitions into transactions reflecting the implementation of an automaton on top of another. Therefore, an anticipation mapping is viewed as a special automaton morphism (a kind of implementation morphism) [7] where the target object is closed under computation, i.e., the target (more concrete) automaton is enriched with all the conceivable sequential and nonsequential computations that can be split into permutations of original transitions. Accordingly, the anticipation of an object is specified over an existing object (an action may be mapped into a complex action of the target object). Also, an action may be mapped according to several alternatives, that is, an anticipation may be state dependent. Thus, we say a more abstract object is "implemented" over a more concrete object, possibly specifying alternative "implementations". In other words, an

CP627, *Computing Anticipatory Systems: CASYS 2001 — Fifth International Conference,* edited by D. M. Dubois

action of the source object may have more than one implementation (possible system anticipations) which may be explicit (alternatives are explicit in the source object) or implicit (actions in the target object used in an anticipation have alternatives). Without alternative implementations, anticipation can be seen as an abstraction mechanism in the language (such as the top-down or bottom-up approach design of systems). In this way, Nautilus is a concurrent programming language in which the objects uses the knowledge of past, present and, in some sense, future states.

2 NONSEQUENTIAL AUTOMATA

Nonsequential automata constitute a categorial semantic domain based on labeled transition system with full concurrency, where restriction and relabelling are functorial and a class of morphisms stands for anticipation. It is a model for concurrency which satisfies the diagonal compositionality requirement, i.e., anticipations compose (vertically) and distribute over combinators (horizontally).

A nonsequential automaton is a special kind of automaton in which states and transitions posses a commutative monoidal structure. A structured transition specifies an independence or concurrency relationship between the component transitions. A structured state can be viewed as a "bag" of local states where each local state can be regarded as a resource to be consumed or produced, like a token in Petri nets.

Nonsequential automata and its morphisms constitute a category which is complete and cocomplete with products isomorphic to coproducts. A product (or coproduct) can be viewed as the parallel composition. In what follows *CMon* denotes the category of commutative monoids and suppose that $i \in I$ where I is a set and $k \in \{0, 1\}$ (for simplicity, we omit that $i \in I$ and $k \in \{0,1\}$).

Definition (Nonsequential Automaton) *A nonsequential automaton* $N = (\boldsymbol{V}, \boldsymbol{T}, \delta_0, \delta_1, \iota, \boldsymbol{L}, lab)$ *is such that* $\boldsymbol{T} = (T, \otimes, \tau)$, $\boldsymbol{V} = (V, \otimes, e)$, $\boldsymbol{L} = (L, \otimes, e)$ *are CMon-objects of transitions, states and labels respectively,* $\delta_0, \delta_1: \boldsymbol{T} \rightarrow \boldsymbol{V}$ *are CMon-morphisms called source and target respectively,* $\iota: \boldsymbol{V} \rightarrow \boldsymbol{T}$ *is a CMon-morphism such that* $\delta_k \circ \iota = id_V$ *and* $lab: \boldsymbol{T} \rightarrow \boldsymbol{L}$ *is CMon-morphism such that* $lab(t) = \tau$ *whenever there is* $v \in \boldsymbol{V}$ *where* $\iota(v) = t$.

Therefore, a nonsequential automaton $N = (\boldsymbol{V}, \boldsymbol{T}, \delta_0, \delta_1, \iota, \boldsymbol{L}, lab)$ can be seen as $N = (G, \boldsymbol{L}, lab)$ where $G = (\boldsymbol{V}, \boldsymbol{T}, \delta_0, \delta_1, \iota)$ is a reflexive graph internal to *CMon* (i.e., $\boldsymbol{V}$, $\boldsymbol{T}$ are *CMon*-objects and δ_0, δ_1, ι are *CMon*-morphisms) representing the automaton shape, $\boldsymbol{L}$ is a comutative monoid representing the labels and lab is the labeling morphism. In an automaton, a transition labeled by τ represents a hidden transition (and therefore, can not be triggered from the outside). Note that, all idle transitions are hidden. The labeling procedure is not extensional in the sense that two distinct transitions with the same label may have the same source and target states (as we will se later, it is essential to give semantics for an object anticipation in Nautilus).

A transition t such that $\delta_0(t) = X$, $\delta_1(t) = Y$ is denoted by $t: X \rightarrow Y$. Since a state is an element of a monoid, it may be denoted as a formal sum $n_1A_1 \oplus ... \oplus n_mA_m$, with the order of the terms being immaterial, where A_i is in $\boldsymbol{V}$ and n_i indicate the multiplicity of the corresponding (local) state, for $i = 1...m$. The denotation of a transition is analogous. We also refer to a structured transition as the parallel composition of component transitions. When no confusion is possible, a structured transition $x \otimes \tau: X \oplus A \rightarrow Y \oplus A$ where $t: X \rightarrow Y$ and $\iota_A: A \rightarrow A$ are labeled by x and τ, respectively, is denoted *by* $x: X \oplus A \rightarrow Y \oplus A$.

Definition (Nonsequential Automaton Morphism) *A nonsequential automaton morphism* $h: N_1 \rightarrow N_2$ *where* $N_1 = (\boldsymbol{V_1}, \boldsymbol{T_1}, \delta_{01}, \delta_{11}, \iota_1, \boldsymbol{L_1}, lab_1)$ *and* $N_2 = (\boldsymbol{V_2}, \boldsymbol{T_2}, \delta_{02}, \delta_{12}, \iota_2, \boldsymbol{L_2}, lab_2)$ *is a triple* $h = (h_V, h_T, h_L)$ *such that* $h_V: \boldsymbol{V_1} \rightarrow \boldsymbol{V_2}$, $h_T: \boldsymbol{T_1} \rightarrow \boldsymbol{T_2}$, $h_L: \boldsymbol{L_1} \rightarrow \boldsymbol{L_2}$ *are CMon-morphisms,* $h_v \circ \delta_{k1} = \delta_{k2} \circ h_T$, $h_T \circ \iota_1 = \iota_2 \circ h_V$ *and* $h_L \circ lab_1 = lab_2 \circ h_T$.

Nonsequencial automata and their morphisms constitute the category *NAut*.

Proposition *The category NAut is bicomplete with products isomorphic to coproducts.*

The category of nonsequential automaton is rich in categorial constructions (restriction, relabelling, synchronization, encapsulation and anticipation) which provides the basic combinators for the Nautilus language described in the next section.

Restriction and relabelling of transitions are functorial operations defined using the fibration and cofibration techniques. Both functors are induced by morphisms at the label level. The restriction operation restricts an automaton "erasing" all those transitions which do not reflect some given table of restrictions:

a) let N be a *NAut*-object with $\boldsymbol{L}$ as the *CMon*-object of labels, ***Table*** be a *CMon*-object, called table of restrictions, and *restr*: ***Table*** $\rightarrow \boldsymbol{L}$ be a restriction morphism. Let $\boldsymbol{u}: NAut \rightarrow CMon$ be the obvious forgetful functor taking each automaton into its labels;

b) the functor $\boldsymbol{u}$ is a fibration and the fibers $\boldsymbol{u^{-1}\ Table}$, $\boldsymbol{u^{-1}\ L}$ are subcategories of *NAut*. The fibration $\boldsymbol{u}$ and the morphism *restr* induce a functor $\boldsymbol{restr}$: $\boldsymbol{u^{-1}\ L \rightarrow u^{-1}\ Table}$. The functor $\boldsymbol{restr}$ applied to *N* provides the automaton reflecting the desired restrictions.

A relabelling relabels the transitions of an automaton according to some morphism of labels. The steps for relabelling are as follows:

a) let *N* be a *NAut*-object with $\boldsymbol{L_1}$ as the *CMon*-object of labels, *relab*: $\boldsymbol{L_1 \rightarrow L_2}$ be a relabelling morphism. Let $\boldsymbol{u}$ be the same forgetful functor used for synchronization purpose;

b) the functor $\boldsymbol{u}$ is a cofibration (and therefore, a bifibration) and the fibers $\boldsymbol{u^{-1}\ L_1}$, $\boldsymbol{u^{-1}\ L_2}$ are subcategories of *NAut*. The cofibration $\boldsymbol{u}$ and the morphism *relab* induce a functor $\boldsymbol{relab}$: $\boldsymbol{u^{-1}\ L_1 \rightarrow u^{-1}\ L_2}$. The functor $\boldsymbol{relab}$ applied to *N* provides the automaton reflecting the desired relabelling.

Proposition *The forgetful functor* $\boldsymbol{u}$: *NAut* $\rightarrow$ *CMon that takes each nonsequential automaton onto its underlying commutative monoid of labels is a fibration and a cofibration.*

Definition (Functor *restr*) *Consider the fibration* $\boldsymbol{u}$: *NAut* $\rightarrow$ *CMon, the automaton automaton* $N = (\boldsymbol{V, T}, \delta_0, \delta_1, \iota, \boldsymbol{L}, lab)$ *and the restriction morphism restr:* $\boldsymbol{Table \rightarrow L}$. *The restriction of N is given by the functor* $\boldsymbol{restr}$: $\boldsymbol{u^{-1}\ L \rightarrow u^{-1}\ Table}$ *induced by* $\boldsymbol{u}$ *and relab applied to N.*

Definition (Functor *relab*) *Consider the fibration* $\boldsymbol{u}$: *NAut* $\rightarrow$ *CMon, the automaton automaton* $N = (\boldsymbol{V, T}, \delta_0, \delta_1, \iota, \boldsymbol{L_1}, lab)$ *and the relabelling morphism relab:* $\boldsymbol{L_1 \rightarrow L_2}$. *The relabelling of N satisfying relab is given by the* $\boldsymbol{relab}$: $\boldsymbol{u^{-1}\ L_1 \rightarrow u^{-1}\ L_2}$ *induced by* $\boldsymbol{u}$ *and relab applied to N.*

Synchronization and encapsulation of nonsequential automata are special cases of restriction and relabelling, respectively. Since the product (or coproduct) construction in *NAut* stands for parallel composition, reflecting all possible combinations between component transitions, it is possible to define a synchronization operation using the restriction operation erasing from the parallel composition all those transition which do not reflect some table of synchronizations (see [8]). A view of an automaton is obtained through hiding of transitions. A hidden transition, i.e. relabelled by τ, cannot be used for synchronization.

2.1 Anticipation in Nonsequential Automata

An anticipation maps transitions into transactions reflecting the implementation of an automaton on top of another. Therefore, an anticipation mapping is viewed as a special automaton morphism (a kind of implementation morphism) where the target object is closed under computation, i.e., the target (more concrete) automaton is enriched with all the conceivable sequential and nonsequential computations that can be split into permutations of original transitions.

In the text that follows, the category of categories internal to *CMon* is denoted by *Cat(CMon)*, *RGr(CMon)* is the category of reflexive graphs internal to *CMon*. We introduce the category L*Cat(CMon)* which can be viewed as a generalization of labeling on *Cat(CMon)*. There is a forgetful functor from L*Cat(CMon)* into *NAut*. This functor has a left adjoint which freely generates a nonsequential automaton into a labeled internal category. The composition of both functors from *NAut* into L*Cat(CMon)* leads to an endofunctor, called transitive closure. The composition of anticipations of nonsequential automata is defined using Kleisli categories (see [9]). In fact, the adjunction above induces a monad which defines a Kleisli category. We show that anticipation distributes over the parallel composition and therefore, the resulting category of automata and anticipations satisfies the diagonal compositionality.

Definition (Category L*Cat(CMon)*) *Consider the category Cat(CMon). The category* L*Cat(CMon) is the comma category* $id_{Cat(CMon)} \downarrow id_{Cat(CMon)}$ *where* $id_{Cat(CMon)}$ *is the identity functor in Cat(CMon).*

Therefore, a L*Cat(CMon)*-object is a triple $\mathsf{N} = (\mathsf{G}, \mathsf{L}, \mathsf{lab})$ where G, L are *Cat(CMon)*-objects and lab is a *Cat(CMon)*-morphism.

Proposition *The category* L*Cat(CMon) has all (small) products and coproducts. Moreover, products and coproducts are isomorphic.*

Definition (Functor *cn*) *Let* $\mathsf{N} = (\mathsf{G}, \mathsf{L}, \mathsf{lab})$ *be a* L*Cat(CMon)-object and* $\mathsf{h} = (\mathsf{h_G}, \mathsf{h_L})$: $\mathsf{N_1} \rightarrow \mathsf{N_2}$ *be a* L*Cat(CMon)-morphism. The functor* $\boldsymbol{cn}$: L*Cat(CMon)* $\rightarrow$ *NAut is such that:*

a) the Cat(CMon)-object $\mathsf{G} = (\boldsymbol{V, T}, \delta_0, \delta_1, \iota, ;)$ *is taken into the RGr(CMon)-object* $G = (\boldsymbol{V, T'}, \delta_0', \delta_1', \iota')$, *where* $\boldsymbol{T'}$ *is* $\boldsymbol{T}$ *subject to the equational rule below and* δ_0', δ_1', ι' *are induced by* δ_0, δ_1, ι *considering the monoid* $\boldsymbol{T'}$; *the*

Cat(CMon)-object $\mathsf{L} = (\boldsymbol{V}, \boldsymbol{L}, \delta_0, \delta_1, \iota, ;)$ *is taken into the CMon-object* $\boldsymbol{L'}$, *where* $\boldsymbol{L'}$ *is* $\boldsymbol{L}$ *subject to the same equational rule; the LCat(CMon)-object* $\mathsf{N} = (\mathsf{G}, \mathsf{L}, \mathsf{lab})$ *is taken into the NAut-object* $N = (G, \boldsymbol{L'}, lab)$ *where lab is the RGr(CMon)-morphism canonically induced by the Cat(CMon)-morphism lab;*

$$\frac{t: A \to B \in T' \quad u: B \to C \in T' \quad t': A' \to B' \in T' \quad u': B' \to C' \in T'}{(t;u) \otimes (t';u') = (t \otimes t');(u \otimes u')}$$

b) the L*Cat(CMon)-morphism* $\mathsf{h} = (\mathsf{h_G}, \mathsf{h_L}): \mathsf{N_1} \to \mathsf{N_2}$ *with* $\mathsf{h_G} = (h_{NV}, h_{NT})$, $\mathsf{h_L} = (h_{LV}, h_{LT})$ *is taken into the NAut-morphism* $h = (h_{NV}, h_{NT'}, h_{LT'}): N_1 \to N_2$ *where* $h_{NT'}$ *and* $h_{LT'}$ *are the monoid morphisms induced by* h_{NT} *and* h_{LT}, *respectively.*

The functor ***cn*** has a requirement about concurrency which is $(t;u)\otimes(t';u') = (t\otimes t');(u\otimes u')$. That is, the computation determined by two independent composed transitions $t;u$ and $t';u'$ is equivalent to the computation whose steps are the independent transitions $t\otimes t'$ and $u\otimes u'$.

Definition (Functor *nc*) *Let* $A = (G, \boldsymbol{L}, lab)$ *be a NAut-object and* $h = (h_G, h_L): A_1 \to A_2$ *be a NAut-morphism. The functor* ***nc****:* $NAut \to$ L*Cat(CMon) is such that:*

a) the RGr(CMon)-object $G = (\boldsymbol{V}, \boldsymbol{T}, \delta_0, \delta_1, \iota)$ *with* $\boldsymbol{V} = (V, \oplus, e)$, $\boldsymbol{T} = (T, \otimes, \tau)$ *is taken into the Cat(CMon)-object* $G = (\boldsymbol{V}, \boldsymbol{T^C}, \delta_0^C, \delta_1^C, \iota, ;)$ *with* $\boldsymbol{T^C} = (T^C, \otimes, \tau)$, δ_0^C, δ_1^C, $\_;\_: \boldsymbol{T^C} \times \boldsymbol{T^C} \to \boldsymbol{T^C}$ *inductively defined as follows:*

$$\frac{t: A \to B \in T}{t: A \to B \in T^C}$$

$$\frac{t: A \to B \in T^C \quad u: B \to C \in T^C}{t;u: A \to C \in T^C} \qquad \frac{t: A \to B \in T^C \quad u: C \to D \in T^C}{t \otimes u: A \oplus C \to B \oplus D \in T^C}$$

subject to the following equational rules:

$$\frac{t \in T^C}{\tau;t = t \;\&\; t;\tau = t} \qquad \frac{t: A \to B \in T^C}{\iota_A;t = t \;\&\; t;\iota_B = t}$$

$$\frac{t: A \to B \in T^C \quad u: B \to C \in T^C \quad v: C \to D \in T^C}{t;(u;v) = (t;u);v}$$

$$\frac{t \in T^C \quad u \in T^C}{t \otimes u = u \otimes t} \qquad \frac{t \in T^C}{t \otimes \tau = t} \qquad \frac{\iota_A \in T^C \quad \iota_B \in T^C}{\iota_A \otimes \iota_B = \iota_{A \oplus B}}$$

$$\frac{t \in T^C \quad u \in T^C \quad v \in T^C}{t \otimes (u \otimes v) = (t \otimes u) \otimes v}$$

the CMon-object $\boldsymbol{L}$ *is taken into the Cat(CMon)-object* $\mathsf{L} = (1, \boldsymbol{L^C}, !, !, !, ;)$ *as above; the NAut-object* $A = (G, \boldsymbol{L}, lab)$ *is taken into the* L*Cat(CMon)-object* $\mathsf{A} = (\mathsf{G}, \mathsf{L}, \mathsf{lab})$ *where lab is the morphism induced by lab;*

b) the NAut-morphism $h = (h_V, h_T, h_L): A_1 \to A_2$ *is taken into the Cat(CMon)-morphism* $\mathsf{h} = (\mathsf{h_G}, \mathsf{h_L}): \mathsf{A_1} \to \mathsf{A_2}$ *where* $\mathsf{h_G} = (h_V, h_{TC})$, $\mathsf{h_L} = (!, h_{LC})$ *and* h_{TC}, h_{LC} *are the monoid morphisms generated by the monoid morphisms* h_T *and* h_{TL}, *respectively.*

Proposition *The functor* ***nc****:* $NAut \to$ L*Cat(CMon) is left adjoint to* ***cn****:* L*Cat(CMon)* $\to NAut$.

Definition (Transitive Closure Functor) *The transitive closure functor is* ***tc*** $=$ ***cn*** o ***nc***: $NAut \to NAut$.

Let $\langle$***nc***, ***cn***, η, $\varepsilon\rangle$: $NAut \to$ L*Cat(CMon)* be the adjunction above. Then, $\mathrm{T} = \langle$***tc***, η, $\mu\rangle$ is a monad on *NAut* such that $\mu =$ ***cn*** ε ***nc***: $\boldsymbol{tc}^2 \to$ ***tc*** where ***cn***: ***cn*** $\to$ ***cn*** and ***nc***: ***nc*** $\to$ ***nc*** are the identity natural transformations and ***cn*** ε ***nc*** is the horizontal composition of natural transformations. For some given automaton N, ***tc*** N is N enriched with its computations, η_{N}: $\mathrm{N} \to$ ***tc*** N includes N into its computations and μ_{N}: $\boldsymbol{tc}^2\mathrm{N} \to$ ***tc*** N flattens computations of computations into computations.

An anticipation morphism φ from A into the computations of B could be defined as a *NAut*-morphism

φ: A $\rightarrow$ ***tc*** B and the composition of anticipations as in Kleisli categories (each monad defines a Kleisli category). However, for giving semantics of objects in Nautilus, anticipations should not preserve labeling (and thus, they are not *NAut*-morphisms). As we show below, each anticipation induces a *NAut*-morphism. Therefore, we may define a category whose morphisms are *NAut*-morphisms induced by anticipations. Both categories are isomorphic.

Definition (Anticipation) *Let $t = \langle \mathbf{tc}, \eta, \mu \rangle$ where $\eta = \langle \eta_G, \eta_L \rangle$, $\mu = \langle \mu_G, \mu_L \rangle$ be the monad induced by the adjunction $\langle \mathbf{nc}, \mathbf{cn}, \eta, \varepsilon \rangle$: NAut $\rightarrow$ L*Cat(CMon). *The category of nonsequential automata and anticipations, denoted by ANAut, is such that (suppose the NAut-objects $N_k = \langle G_k, \mathbf{L}_k, lab_k \rangle$, for k in {1, 2, 3}):*

a) *ANAut-objects are the NAut-objects*;

b) *$\varphi = \varphi_G$: $N_1 \rightarrow N_2$ is a ANAut-morphism where φ_G: $G_1 \rightarrow \mathbf{tc}\, G_2$ is a RGr(CMon)-morphism and for each NAut-object N, $\varphi = \eta_G$: $N \rightarrow N$ is the identity morphism of N in ANAut*;

c) *let φ: $N_1 \rightarrow N_2$, ψ: $N_2 \rightarrow N_3$ be ANAut-morphisms. The composition ψ o φ is a morphism $\psi_G \, o_K \, \varphi_G$: $N_1 \rightarrow N_3$ where $\psi_G \, o_K \, \varphi_G$ is as illustrated in Figure 1.*

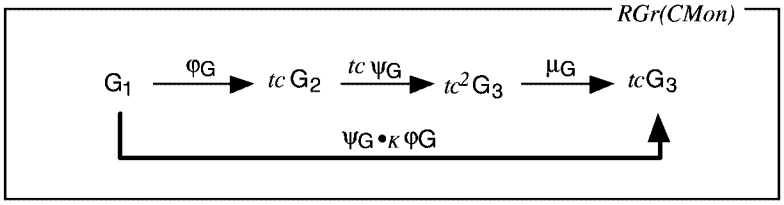

FIGURE 1. Composition of Anticipations

In what follows, an automaton $\langle G, \mathbf{L}, lab \rangle$ may be denoted as a morphism lab: $G \rightarrow inc\ \mathbf{L}$ or it is abbreviated just by lab: $G \rightarrow \mathbf{L}$.

Definition (Anticipation with Induced Labeling) *Let $t = \langle \mathbf{tc}, \eta, \mu \rangle$ where $\eta = \langle \eta_G, \eta_L \rangle$, $\mu = \langle \mu_G, \mu_L \rangle$ be the monad induced by the adjunction $\langle \mathbf{nc}, \mathbf{cn}, \eta, \varepsilon \rangle$. The category of nonsequential automata and anticipations with induced labeling $ANAut_L$ is such that (suppose the NAut-objects $N_k = \langle G_k, \mathbf{L}_k, lab_k \rangle$, for k in {1, 2, 3}):*

a) $ANAut_L$-objects are the NAut-objects;

b) let φ_G: $G_1 \rightarrow \mathbf{tc}\, G_2$ be a RGr(CMon)-morphism. Then $\varphi = \langle \varphi_G, \varphi_L \rangle$: $N_1 \rightarrow N_2$ is a $ANAut_L$-morphism where φ_L is given by the pushout illustrated in the Figure 2 (left). For each NAut-object N, $\varphi = \langle \eta_G$: $G \rightarrow \mathbf{tc}\, G$, φ_L: $\mathbf{L} \rightarrow \mathbf{L}_\eta \rangle$: $N \rightarrow N$ is the identity morphism of N in $ANAut_L$ where φ_L is as above;

c) let φ: $N_1 \rightarrow N_2$, ψ: $N_2 \rightarrow N_3$ be $ANAut_L$-morphisms. The composition ψ o φ is a morphism $\langle \psi_G \, o_K \, \varphi_G$, $\psi_L \, o_L \, \varphi_L \rangle$: $N_1 \rightarrow N_3$ where $\psi_G \, o_K \, \varphi_G$ e $\psi_L \, o_L \, \varphi_L$ is as illustrated in Figure 2 (right).

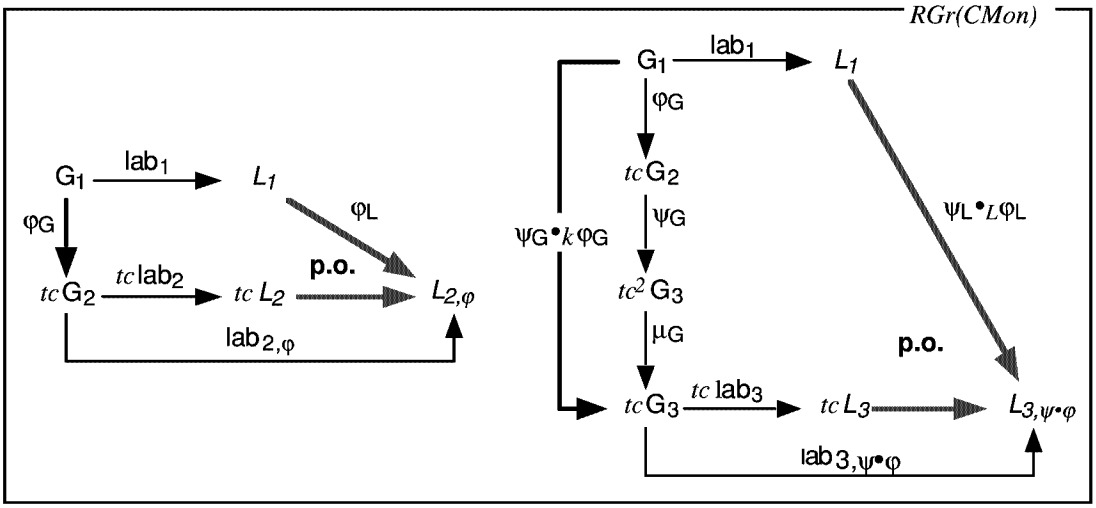

FIGURE 2. Induced Labelling in Anticipation

It is easy to prove that *ANAut* and $ANAut_L$ are isomorphic (we identify both categories by *ANAut*). Thus, every anticipation morphism can be viewed as a *NAut*-morphism. For a *ANAut*-morphism φ: A $\rightarrow$ B, the corresponding *NAut*-morphism is denoted by φ: A $\rightarrow$ ***tc*** B.

Since anticipations constitute a category, the vertical compositionality is achieved. In the following proposition, we show that, for some given anticipation morphisms, the morphism (uniquely) induced by the parallel composition is also an anticipation morphism and thus, the horizontal compositionality is (also) achieved.

Proposition *Let* $\{\varphi_i: N1_i \rightarrow \mathbf{tc}\, N2_i\}$ *be an indexed family of anticipations. Then* $\times_{i\in I}\, \varphi_i: \times_{i\in I}\, N1_i \rightarrow \times_{i\in I}\, \mathbf{tc}\, N2_i$ *is an anticipation.*

2.2 Restriction and Relabeling of Anticipations

The restriction of an anticipation is the restriction of the source automaton. The restriction of a community of anticipations (i.e., the parallel composition of anticipated automata) is the restriction of the parallel composition of the source automata whose anticipation is induced by the component anticipations. Note that, in the following construction, we assume that the horizontal compositionality requirement is satisfied. Remember that ***tc*** preserves products and that every restriction morphism has a cartesian lifting at the automata level.

Definition (Restriction of an Anticipation) *Let* $\varphi: N1 \rightarrow \mathbf{tc}\, N2$ *be an anticipation and* $restr_L$: ***Table*** $\rightarrow \mathbf{L}_1$ *be a restriction morphism and* $restr_N$: ***restr*** $N1 \rightarrow N1$ *be its cartesian lifting. The anticipation of the restricted automaton* ***restr*** $N1$ *is* ***restr*** φ: ***restr*** $N1 \rightarrow \mathbf{tc}\, N2$ *such that* ***restr*** $\varphi = \varphi\, o\, restr_N$.

Proposition *Let* $\{\varphi_i: N1_i \rightarrow \mathbf{tc}\, N2_i\}$ *be an indexed family of anticipations where* $N_{ki} = \langle G_{ki}, \mathbf{L}_{ki}, lab_{ki}\rangle$. *Let* $restr_L$: ***Table*** $\rightarrow \times_i \mathbf{L1}_i$ *be a restriction morphism and* $restr_N$: ***restr*** $N1_i \rightarrow \times_i N1_i$ *be its cartesian lifting. The restriction of the parallel composition of component anticipations is* ***restr*** φ_i: ***restr*** $N1_i \rightarrow \mathbf{tc}\, (\times_i N2_i)$ *such that* ***restr*** $\varphi_i = \times_i\, \varphi_i\, o\, restr_N$ *where* $\times_i\, \varphi_i$ *is uniquely induced by the product construction.*

The relabelling of an anticipation is induced by the relabelling of the source automaton.

Definition (Relabelling of an Anticipation) *Consider the Figure 3. Let* $\varphi: N1 \rightarrow \mathbf{tc}\, N2$ *be an anticipation where* $N_k = \langle G_k, \mathbf{L}_k, lab_k\rangle$ *and* $\varphi = \langle \varphi_G, \varphi_L\rangle$. *Let lab:* $\mathbf{L}_1 \rightarrow \mathbf{L}_1'$ *be a relabelling morphism and* ***relab*** $N1 = \langle G1, \mathbf{L}_1', relab\ o\ lab_1\rangle$ *the relabelled automaton. Then, the relabelling of the anticipation morphism is* ***relab*** $\varphi = \langle \varphi_G,$ ***relab*** $\varphi_L\rangle$.

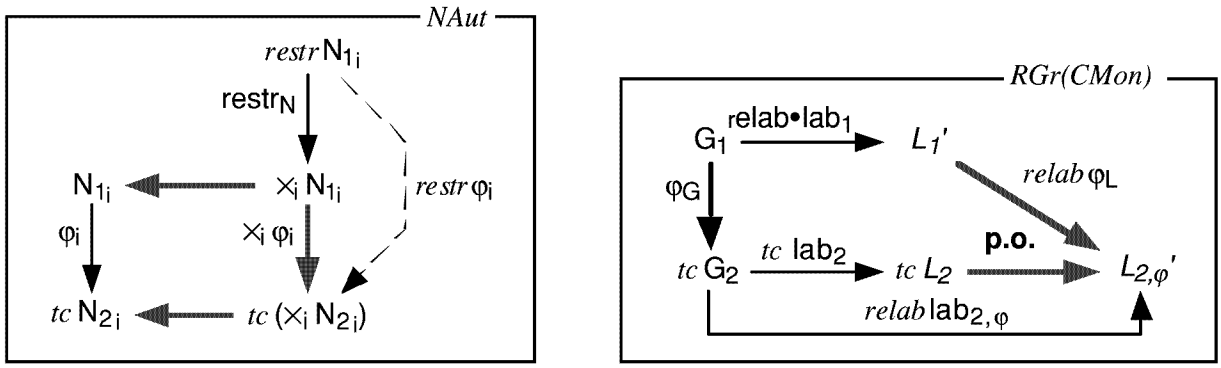

FIGURE 3. Restriction and Relabelling of Anticipations

3 NAUTILUS: ITS SYNTAX AND SEMANTICS

The language named Nautilus is based on the object-oriented language GNOME [11] which is a simplified and revised version of OBLOG [12, 13, 14]. It is a high level specification language for concurrent, communicating systems. The main features of Nautilus are the following:

- objects may interact through callings;
- an object may be the aggregation of component objects;

- an object may be anticipated into a sequential or parallel computations of another object;
- an object may be a view of another object;
- interaction, aggregation and anticipation may be state-dependent, i.e., may depend dynamically on some conditions;
- interaction, aggregation and anticipation are compositional;
- the evaluation of an action is atomic;
- the clauses of an action may be composed in a sequential or multiple ways.

In this brief discussion of the language Nautilus we introduce some key words in order to help the understanding of the examples below. The specification of an object in Nautilus depends on if it is a simple object or a structured object such as an anticipation (`over`) or a parallel composition. In any case, a specification has two main parts: interface and body. The interface declares the category (`category`) of some actions (`birth, death`). The body (`body`) declares the attributes (`slot` - only for the simple object) and the methods of all actions. A birth or death action may occur at most one time (and determines the birth or the death of the object). An action may occur if its enabling (`enb`) condition holds. An action with alternatives (`alt`) is enabled if at least one alternative is enabled. In this case, only one enabled alternative may occur where the choice is an internal nondeterminism. The evaluation of an action (or an alternative within an action) is atomic. An action may be a sequential (`seq/end seq`) or multiple (`cps/end cps`) composition of clauses. A multiple composition is a special composition of concurrent clauses based on Dijkstra's guarded commands [4] where the valuation (`val`) clauses are evaluated before the results are assigned to the corresponding slots. Due to space restrictions, we introduce some details of the language Nautilus through examples (in the next section) and, at the same time, we give its semantics using nonsequential automata.

A nonsequential automata semantics for Nautilus is easily defined. Since an action may be a sequential or multiple composition of clauses executed in an atomic way, the semantics of a simple object is given by an anticipation morphism where the target automaton called base is determined by the computations of a freely generated automaton able to simulate any object specified over the involved attributes and the source automaton is a relabelled restriction of the base. Therefore, the semantics of an action in Nautilus is a nonsequential automaton transition (and thus is atomic) anticipated into a (possible complex) computation.

The semantics of a anticipation is the composition of semantics, i.e., the anticipation of the source automata over the target composed with the anticipation of the target over its base.

The semantics of an interaction, aggregation and encapsulation in Nautilus are straightforward since they are given by synchronization and encapsulation of anticipation morphisms of nonsequential automata (an aggregation also defines a relabelling). An action with inputs or outputs is associated to a family of transitions indexed by the corresponding values.

The semantics of a community of concurrent objects is the parallel composition of the semantics of component objects, i.e., the parallel composition of anticipations.

4 NAUTILUS AND ANTICIPATION

In this section we present some examples depicting the features of the Nautilus language. The examples are all presented in textual format instead of the visual diagrammatic one for purpose of simplicity and to keep the paper short.

We borrow the example called "The Carrot Principle" from Dubois [5] in order to introduce the anticipation features, syntax and semantics of the Nautilus language. In this example "You like to go for a donkey ride. You sit on the donkey which does not want to go. So you present a carrot before it and then it goes to capture the carrot, but at the same time the carrot goes also before it. The carrot is an "anticipatory attractor" of the movement. To go to the left or to the right, you position the carrot to the left or to the right respectively: this defines a selection of one particular trajectory amongst all the potential possible subtrajectories." As mentioned by Dubois, this example depicts two kinds of anticipation: the attractor which is the motor of the action, and several selections of local anticipatory trajectories aiming at obtaining a global anticipatory trajectory or the final goal.

We have specified in Nautilus other examples of objects for systems with anticipation behavior. We can mention among others which are not presented here due to limitation of space: two players competing against each other in a game, a simulation of a simple case of free-will, a system which tries to avoid unwanted future states, etc.

The Nautilus code for the "Carrot Principle" is shown in Tables 1, 2, 3, 4. As the purpose of this example is to explore the building blocks of the language, it is not necessarily the best example of specification one could built using Nautilus. It has two simple objects named *Donkey* and *Carrot*, which specifies the behavior of the basic

objects that participate in our system. An aggregation of these two simple object is defined as the *Guided_Donkey* object, which behavior corresponds to the synchronization of the *Donkey* and the *Carrot* (it describes the "anticipatory attractor" of the donkey movement). Finally we define a possible anticipation between all the possible paths the *Guided_Donkey* may fallow, thus building the *Trained_Donkey*.

TABLE 1. Specification of Simple Object *Donkey*

```
object  Donkey
export
  New
  Left
  Right
  Walk
  Stop
category
  birth request New
body
  slot Direction: (N,S,E,W)
  slot Action: 0..1
act New
    alt New1
      seq
        val Direction << N
        val Action << 0
      end seq
    alt New2
      cps
        val Direction << S
        val Action << 0
      end cps
    ...
    alt New4
    ...
act Left
    alt Left1
      enb Direction = N
      val Direction << W
    ...
    alt Left4
      enb Direction = W
      val Direction << S
  act Right
    alt Right1
      enb Direction = N
      val Direction << E
    ...
    alt Right4
      enb Direction = W
      val Direction << N
  act Walk
    val Action << 1
  act Stop
    enb Action = 1
    val Action << 0
end Donkey
```

TABLE 2. Specification of Simple Object *Carrot*

```
object Carrot
export
  New
  Left
  Right
  Forward
  Hide
category
  birth request New
body
  slot Pos: (L,R,F,H)
  act New
    val Pos << H
  act Left
    val Pos << L
  act Right
    val Pos << R
  act Forward
    val Pos << F
act Hide
    alt Hide1
      enb Pos = L
      val Pos << H
    alt Hide2
      enb Pos = R
      val Pos << H
    alt Hide3
      enb Pos = F
      val Pos << H
end Carrot
```

TABLE 3. Specification of Aggregation Object *Guided_Donkey*

```
object Guided_Donkey
aggregation of
  Donkey
  Carrot
export
  New
  TurnLeft
  TurnRight
  Walk
  Stop
category
  birth request New
body
  act New composed by
    New of Donkey
    New of Carrot
  act TurnLeft composed by
    Left of Donkey
    Left of Carrot
act TurnRight composed by
    Right of Donkey
    Right of Carrot
  act Walk composed by
    Walk of Donkey
    Forward of Carrot
  act Stop composed by
    Stop of Donkey
    Hide of Carrot
end Guided_Donkey
```

TABLE 4. Specification of Anticipation Object *Trained_Donkey*

object Trained_Donkey over Guided_Donkey export Born Go Return category birth request Born request Go request Return body act Born New act Go alt Go1	seq Walk TurnLeft Walk Turn Right Walk end seq alt Go2 seq TurnLeft Walk TurnRight Walk Walk end seq	act Return seq TurnLeft TurnLeft Walk Walk TurnLeft Walk end seq end Trained_Donkey

The *Donkey* is composed of five possible actions: *new* (causes the object to be born, as pointed by the `birth` clause), *left* (the donkey turns left), *right* (the donkey turns right), *walk* (the donkey goes forward), *stop* (the donkey stops moving). Those actions alter the internal state of the object, i.e., they change the values of the object `slots` which possible values are the cardinal points (*N*,*S*,*E*,*W*) for the *direction* slot, and *0* (donkey is not moving) or *1* (donkey is moving) for the *action* slot. The set of actions listed in the sentence `export` are the only actions that can be referenced by other objects. It is important to explain how actions are executed. Active actions are the ones for which every condition regarding to its requirements of usage are satisfied and so that every call can be executed. The selection of the activated action that will arise is an internal nondeterminism. Note that the birth action *new* has four alternatives. Both alternatives are always enabled, since they do not have enabling conditions. However, since it is a birth action, it occurs only once. For the purpose of explaining the semantics of Nautilus we have used the sequential (`seq`) composition of clauses in the alternative *new1* and the multiple (`cps`) composition in the other alternatives.

The semantics of an independent object in Nautilus is given by an anticipation morphism of nonsequential automata as follows:

- The target automata called base is determined by the computations of a freely generated automata able to simulate any object specified over the involved attributes, i.e., it is defined as the computations of an automaton whose states are freely generated by the set of all possible values of all slots and the transitions are freely generated by the set of all possible transitions between values of component attributes.
- From the nonsequential automata computational closure, every possible computation over the object attributes may be simulated. In this sense, the target automata anticipates all computation paths for the specified object.
- The source automaton is a relabelled restriction of the base. The restriction can be seen as the operation which selects the desired paths from all the possible anticipations.
- Finally, the anticipation morphism Donkey: D1→*tc*D2, where D1 and D2 are nonsequential automata, is partially represented in Figure 4. In the figure we show the mappings for two of the four alternatives of the action *New* and also for the *Stop* action and only one of the *Left* actions. In the target automaton, the bold lines represent part of the automaton used in the restriction morphism. Detailed instructions on how the morphisms and the automata are built can be found in [8].

The behavior of the object *Carrot* is analogous and its semantics can be given in a similar fashion.

Aggregation consists in synchronizing two objects, therefore in a deeper level, synchronizing two automata. In the example, the object *Guided_Donkey* is implemented as the aggregation (`aggregation of`) of two other objects *Donkey* and *Carrot* respectively. The initially separated parts are now faced together forming an aggregator object that will specify the aggregated objects behaviors. The semantics of an aggregation is the result of the synchronization of anticipations of nonsequential automata.

The structured object *Trained_Donkey* is constructed using the anticipation clause (`over`) in Nautilus. The anticipation constructor in Nautilus aims to set an object in function of the possible transactions from another object, which is called the anticipation base. Thus, the anticipation of an object is specified over an existing object (the *Trained_Donkey* is specified over the *Guided_Donkey*). An action may be anticipated into a complex action (a sequential or multiple composition of clauses) of the target object. In the example, the actions *Go* and *Return* are sequential compositions (`seq/end seq`) and the action *Born* is anticipated into a single action. Also, an action

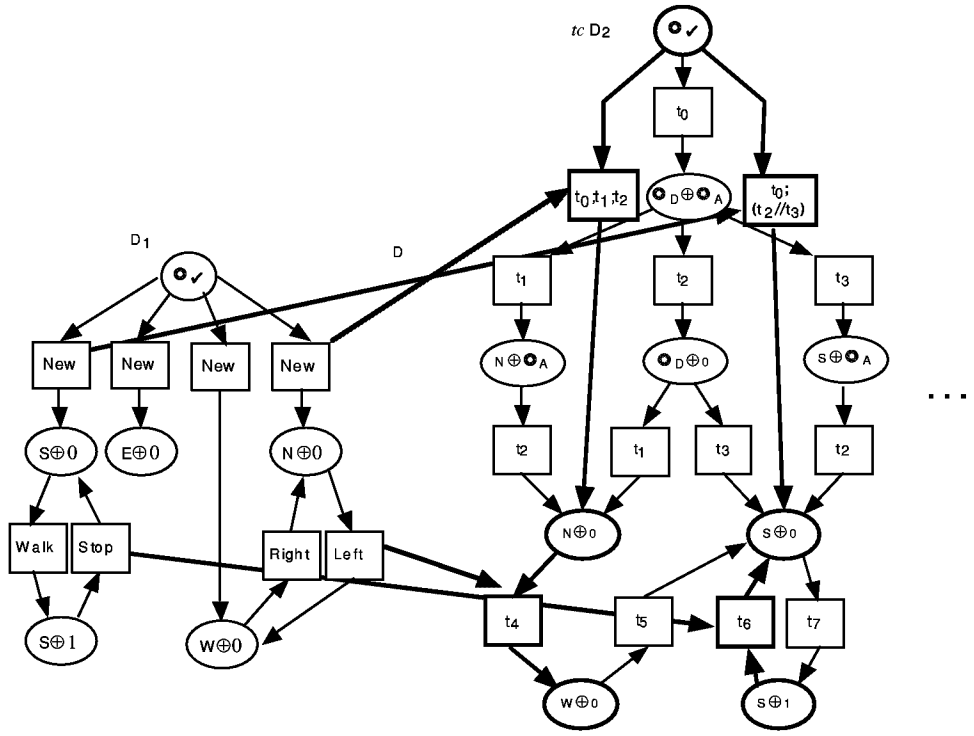

FIGURE 4. Semantics of an Object in Nautilus

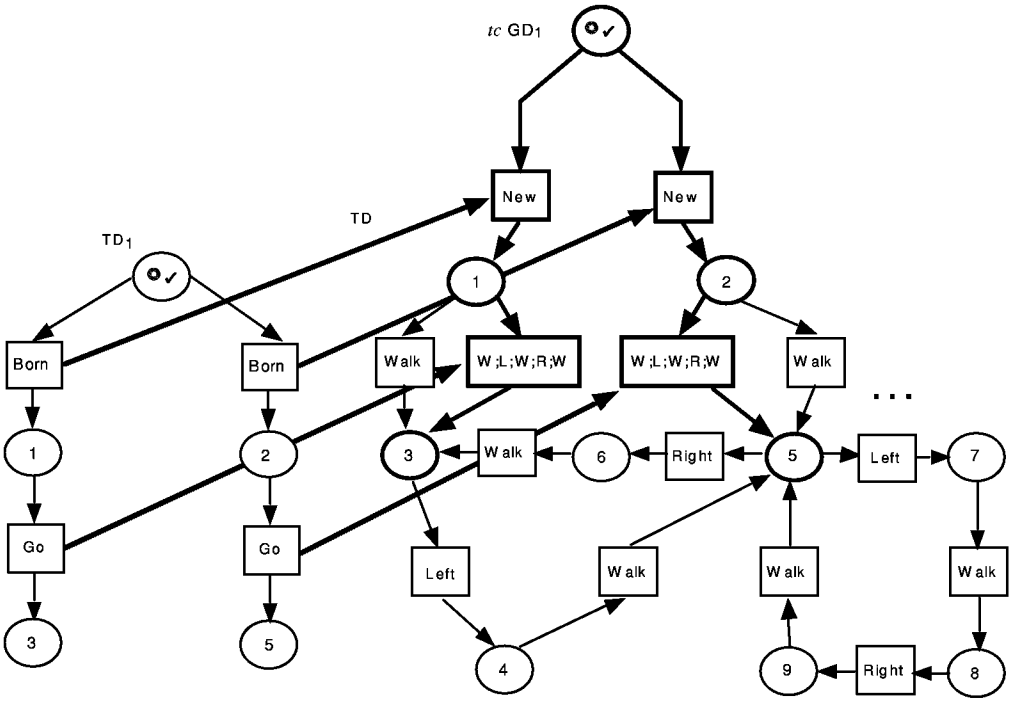

FIGURE 5. Semantics of an anticipation in Nautilus

may be anticipated according to several alternatives, that is, an anticipation may be state dependent. In this sense, an action of the source object may have more then one implementation which may be explicit, i.e. alternatives are explicit in the source object (the action *Go* has two explicit alternatives) or implicit, i.e. actions in the target object used in an anticipation have alternatives (the action *New* has four alternatives as defined in the *Donkey* object). Note that anticipations are compositional and therefore, the target object of an anticipation may be the source of another anticipation.

The semantics of an anticipation is a composition of anticipations, i.e., the anticipation of the source automata over the target composed with the anticipation of the target over its base. Consider the anticipation of the "Carrot Principle" example. Its semantics is given by the anticipation morphism partially illustrated in Figure 5. Let TD (*Trained_Donkey*) be the anticipated object over the GD (*Guided_Donkey*) object. Let GD: GD1→*tc*GD2 the semantics of GD. The semantics of TD is the composition of the corresponding semantics: GDoTD, such that TD: TD1→*tc*GD1, where the nonsequential automaton TD1 is a relabelled restriction of *tc*GD1. Notice we only depict a small part of the nonsequential automata in the figure, showing two implicit anticipations of the *Born* action and two explicit ancitipations of the *Go* action.

5 CONCLUDING REMARKS

The language Nautilus is a concurrent object oriented language, which is based on the language GNOME, and introduces some special features such as anticipation.

In Nautilus, an object can be specified either as a simple object or the resulting object of an encapsulation, aggregation, anticipation or parallel composition. A semantics for Nautilus is given by Nonsequencial automata which constitute a categorial semantic domain with full concurrency, based on structured labeled transition systems, which satisfies the diagonal compositionality requirement, i.e., anticipation compose (vertically) and distributes through the parallel composition (horizontally).

The semantics of a simple object is given by an anticipation morphism where the target automaton called base is determined by the computations of a freely generated automaton able to simulate any object specified over the involved attributes and the source automaton is a relabelled restriction of the base. Therefore, the semantics of an action in Nautilus is a nonsequential automaton transition (and thus is atomic) anticipated into a (possible complex) computation. Also, those computations may have alternatives representing possible system anticipations. The semantics of a anticipation is the composition of semantics, i.e., the anticipation of the source automata over the target composed with the anticipation of the target over its base (anticipation is explained using Kleisli categories).

As a general purpose programming language, Nautilus can be used to specify several different kinds of systems. In this paper we have only covered simple examples depicting the anticipation constructor in Nautilus (which aims to set an object in function of the possible transactions from another object). Nautilus has already been compared to other object-oriented languages, such as Java, and has shown to posses several good features in the specification of data types (e.g. a stack) and complex systems (e.g. a vending machine or a distributed white-board system) (see [2, 3] for a comparison).

For our knowledge, Nautilus is the first general purpose concurrent programming language to include anticipation as a feature.

REFERENCES

1. Asperti, A., and Longo, G., *Categories, Types and Structures - An Introduction to the Working Computer Science*, MIT Press, Cambridge, 1991.
2. Carneiro, C., Reis, R. Q., and Menezes, P. B., "Processamento Concorrente em Nautilus e Java" in *Proceedings of III Brazilian Symposium on Programming Languages-1999*, edited by M. L. Lisboa, SBC, Porto Alegre, 1999, pp. 155-169.
3. Carneiro, C., Veit, T., D' Andrea, F., and Menezes, P. B., "Nautilus: its Concurrent and Distributed Characteristics and as an Academic Language" in *International Conference on Parallel and Distributed Processing Techniques and Applications-1999*, edited by H. R. Arabnia, C.S.R.E.A., Las Vegas, 1999, pp. 1919-1925.
4. Dijkstra, E. W., *A Discipline of Programming*, Prentice Hall, Englewood Cliffs, 1976.
5. Dubois, D., "Review of Incursive, Hyperincursive and Anticipatory Systems – Foundation of Anticipation in Electromagnetism" in *Computing Anticipatory Systems: CASYS' 99 - Third International Conference*, edited by D. M. Dubois, published by The American Institute of Physics, New York, AIP Conference Proceedings 517, 2000, pp. 3-30.
6. Menezes, P. B., and Costa, J. F., *Journal of the Brazilian Computer Society* **2(1)**, 50-67 (1995).

7. Menezes, P. B., Costa, J. F., and Sernadas, A., "Refinement Mapping for (Discrete event) System Theory" in *Computer Aided System Theory - EUROCAST'95*, F. Pichler, R. Moreno Diaz, R. Albrecht (Eds.), Lecture Notes in Computer Science 1030, Springer-Verlag, Berlin, 1996, pp. 103-116.
8. Menezes, P. B., Costa, J. F., and Sernadas, A., "Nonsequencial Automata Semantics for a Concurrent, Object Based Language" in *US-Brazil Joint Workshop on the Formal Foundations of Software Systems -1998*, edited by R. Cleaveland et al., Electronic Notes in Theoretical Computer Science 14, Elsevier Science, Amsterdan, 2000, pp. 29.
9. Menezes, P. B., "A Categorical Framework for Concurrent Anticipatory Systems" in *Computing Anticipatory Systems: CASYS' 98 - Second International Conference*edited by D. M. Dubois, published by The American Institute of Physics, New York, AIP Conference Proceedings 465, 1999, pp. 185-199.
10. Ramos, J., and Sernadas, A., *A Brief Introduction to GNOME*, technical report, Universidade Técnica de Lisboa, Instituto Superior Técnico, Lisboa, 1995. http://www.cs.math.ist.utl.pt/cs/lcg/gnome.html
11. Sernadas, A., and Ramos, J., *A Linguagem GNOME: Sintaxe, Semântica e Cálculo*, technical report, Universidade Técnica de Lisboa, Instituto Superior Técnico, Lisboa, 1994. http://www.cs.math.ist.utl.pt/cs/lcg/gnome.html
12. Sernadas, C., Resende, P., Gouveia, P., and Sernadas, A., "In-the-large Object-Oriented Design of Information Systems" in *The Object-Oriented Approach in Information Systems*, edited by F. van Assche et al., North-Holland, Amsterdan, 1991, pp. 209-232.
13. Sernadas, C., Gouveia, P., Ramos, J., and Resende, P., "The Refinement Dimension in Object-Oriented Database Design" in *Specification of Data Base Systems*, edited by D. Harper and M. Norrie, Springer-Verlag, Berlin, 1992, pp. 275-299.
14. Sernadas, C., Gouveia, P., and Sernadas, A., *OBLOG: Object-Oriented, Logic-Based Conceptual Modeling*, technical report, Universidade Técnica de Lisboa, Instituto Superior Técnico, Lisboa, 1994. http://www.cs.math.ist.utl.pt/cs/lcg/gnome.html