# Graph Transformation with Time: Causality and Logical Clocks

Szilvia Gyapay[1], Reiko Heckel[2], and Dániel Varró[1]

[1] Dept. of Measurement and Information Systems,
Budapest University of Technology and Economics
H-1521 Budapest, Hungary
{gyapay,varro}@mit.bme.hu
[2] Dept. of Math. and Comp. Science, University of Paderborn
D-33095 Paderborn, Germany
reiko@upb.de

**Abstract.** Following TER nets, an approach to the modelling of time in high-level Petri nets, we propose a model of time within (attributed) graph transformation systems where logical clocks are represented as distinguished node attributes. Corresponding axioms for the time model in TER nets are generalised to graph transformation systems and semantic variations are discussed. They are summarised by a general theorem ensuring the consistency of temporal order and casual dependencies.
The resulting notions of *typed graph transformation with time* specialise the algebraic double-pushout (DPO) approach to typed graph transformation. In particular, the concurrency theory of the DPO approach can be used in the transfer of the basic theory of TER nets.

## 1 Introduction

Recently, a number of authors have advocated the use of graph transformation as a semantic framework for visual modelling techniques both in computer science and engineering (see, e.g., the contributions in [3, 4]). In many such techniques, the modelling of time plays a relevant role. In particular, techniques for embedded and safety critical systems make heavy use of concepts like timeouts, timing constraints, delays, etc., and correctness with respect to these issues is critical to the successful operation of these systems. At the same time, those are exactly the systems where, due to the high penalty of failures, formally based modelling and verification techniques are most successful. Therefore, neglecting the time aspect in the semantics of visual modelling techniques, we disregard one of the crucial aspects of modelling.

So far, the theory of graph transformation provides no support for the modelling of time in a way which would allow for *quantified* statements like "this action takes 200ms of time" or "this message will only be accepted within the next three seconds", etc. However, from a more abstract, *qualitative* point of view we can speak of temporal and causal ordering of actions thus abstracting from actual clock and timeout values. Particularly relevant in this context is the theory of concurrency of graph transformation, see [1, 6, 14] or [2] for a recent survey.

It is the objective of this paper to propose a quantitative model of time within graph transformation which builds on this more abstract qualitative model. Therefore, we will not add time concepts on top of an existing graph transformation approach, but we show how, in particular, typed graph transformation systems in the double-pushout (DPO) approach [6] can be extended *from within* with a notion of time. This allows both the straightforward transfer of theoretical results and the reuse of existing tools.

In a recent preliminary paper [10], we have already outlined our approach, proposing several alternative definitions and discussing their consequences with respect to the existence of a globally time-ordered sequence of transformations. Such property is desirable because it witnesses the consistency of time values attached to vertices with the causal dependencies between steps. In this paper, we refine the alternatives discussed in [10] and prove a general condition on graph transformation systems with time ensuring the desired consistency property, thus effectively solving the problem posed in [10] in the most general case.

The following section outlines our general approach of the problem, which is motivated by a corresponding development in Petri nets, briefly to be reviewed in Section 3. Graph transformation with time is introduced and investigated in Section 4 while Section 5 concludes the paper.

## 2 From Nets to Graph Transformation, with Time

When trying to incorporate time concepts into graph transformation, it is inspiring to study the representation of time in Petri nets. Nets are formally and conceptually close to graph transformation systems which allows for the transfer of concepts and solutions. This has already happened for relevant parts of the concurrency theory of nets which, as mentioned above, provides a qualitative model of time based on the causal ordering of actions.

In particular, we will follow the approach of time ER nets [11]. These are simple high-level nets which introduce time as a distinguished data type. Then, time values can be associated with individual tokens, read and manipulated like other token attributes when firing transitions. In order to ensure meaningful behaviour (like preventing time from going backwards) constraints are imposed which can be checked for a given net. The advantage of this approach with respect to our aims is the fact that time is modelled within the formalism rather than adding it on top as a new formal concept.

Based on the correspondence of Petri nets and (typed) graph transformation, which regards Petri nets as rewriting systems on multi-sets of vertices [5], we can derive a model of time within typed graph transformation systems with attributes. The correspondence is visualised in Table 1. Besides (low-level) place-transition nets and typed graph transformation systems, it relates (high-level) environment-relationship nets to typed graph transformation with attributes. This relationship, which has first been observed in the case of algebraic high-level nets [8] and attributed graph transformation [17] in [18], shall enable us to transfer the modelling of time in time ER nets to typed graph transformation with attributes.

Next, we review time environment-relationship (TER) nets [11] in order to prepare for the transfer to typed graph transformation systems in Section 4.

**Table 1.** Corresponding Petri net and graph transformation variants

| | Petri nets | graph transformation systems |
|---|---|---|
| low-level | PT nets | typed graph transformation (TGT) |
| high-level | ER nets | typed graph transformation with attributes (TGTA) |
| with time | TER nets | typed graph transformation with time (TGTT) |

## 3 Modelling Time in Petri Nets

There are many proposals for adding time to Petri nets. In this paper we concentrate on one of them, time ER nets [11], which is chosen for its general approach of considering time as a token attribute with particular behaviour, rather than as an entirely new concept. As a consequence, time ER nets are a special case of ER nets.

### 3.1 ER nets

ER (environment-relationship) nets are high-level Petri nets (with the usual net topology) where tokens are environments, i.e., partial functions $e : ID \to V$ associating attribute values from a given set $V$ to attribute identifiers from a given set $ID$. A marking $m$ is a multi-set of environments (tokens).

To each transition $t$ of the net with pre-domain $p_1 \dots p_n$ and post-domain $p'_1 \dots p'_m$, an action $\alpha(t) \in Env^n \times Env^m$ is associated. The projection of $\alpha(t)$ to the pre-domain represents the firing condition, i.e., a predicate on the tokens in the given marking which controls the enabledness of the transition. If the transition is enabled, i.e., in the given marking $m$ there exist tokens satisfying the predicate, the action relation determines possible successor markings.

Formally, a transition $t$ is enabled in a marking $m$ if there exists a tuple $\langle pre, post\rangle \in \alpha(t)$ such that $pre \leq m$ (in the sense of multiset inclusion). Fixing this tuple, the successor marking $m'$ is computed, as usual, by $m' = (m - pre) + post$, and this firing step is denoted by $m[t(pre, post)\rangle m'$. A firing sequence of $s = m_0[t_1(pre_1, post_1)\rangle \dots [t_{k-1}(pre_{k-1}, post_{k-1})\rangle m_k$ is just a sequence of firing steps adjacent to each other.

### 3.2 Time ER nets

Time is integrated into ER nets by means of a special attribute, called *chronos*, representing the time of creation of the token as a time stamp. Constraints on the time stamps of both (i) given tokens and (ii) tokens that are produced can be specified by the action relation associated to transitions. To provide a meaningful model of time, action relations have to satisfy the following axioms with respect to *chronos* values [11].

**Axiom 1: Local monotonicity** For any firing, the time stamps of tokens produced by the firing can not be smaller than time stamps of tokens removed by the firing.

**Axiom 2: Uniform time stamps** For any firing $m[t(pre, post)\rangle m'$ all time stamps of tokens in $post$ have the same value, called the *time of the firing*.

**Axiom 3: Firing sequence monotonicity** For any firing sequence $s$, firing times should be monotonically nondecreasing with respect to their occurrence in $s$.

The first two axioms can be checked locally based on the action relationships of transitions. For the third axiom, it is shown in [11] that every sequence $s$ where all steps satisfy Axioms 1 and 2 is *permutation equivalent* to a sequence $s'$ where also Axiom 3 is valid. Here, permutation equivalence is the equivalence on firing sequences induced by swapping independent steps. Thus, any firing sequence can be viewed as denoting a representative, which satisfies Axiom 3.

It shall be observed that TER nets are a proper subset of ER nets, i.e., the formalism is not extended but specialised. Next, we use the correspondence between graph transformation and Petri nets to transfer this approach of adding time to typed graph transformation systems.

## 4 Modelling Time in Graph Transformation Systems

Typed graph transformation systems provide a rich theory of concurrency generalising that of Petri nets [2]. In order to represent time as an attribute value, a notion of typed graph transformation with attributes is required. We propose an integration of the two concepts (types and attributes) which presents attribute values as vertices and attributes as edges, thus formalising typed graph transformation with attributes as a special case of typed graph transformation.

Next, we give a light-weight (set-theoretic) presentation of the categorical DPO approach [9] to the transformation of typed graphs [6].

### 4.1 Typed graph transformation

In typed graph transformation, graphs occur at two levels: the type level and the instance level [6]. A fixed *type graph* $TG$ (which may be thought of as an abstract representation of a class diagram) determines a set of *instance graphs* $\langle G, g : G \to TG \rangle$ which are equipped with a structure-preserving mapping $g$ to the type graph (formally expressed as a *graph homomorphism*).

A *graph transformation rule* $p : L \to R$ consists of a pair of $TG$-typed instance graphs $L, R$ such that the union $L \cup R$ is defined. (This means that, e.g., edges which appear in both $L$ and $R$ are connected to the same vertices in both graphs, or that vertices with the same name have to have the same type, etc.) The left-hand side $L$ represents the pre-conditions of the rule while the right-hand side $R$ describes the post-conditions.

A *graph transformation* from a pre-state $G$ to a post-state $H$, denoted by $G \overset{p(o)}{\Longrightarrow} H$, is given by a graph homomorphism $o : L \cup R \to G \cup H$, called *occurrence*, such that

- $o(L) \subseteq G$ and $o(R) \subseteq H$, i.e., the left-hand side of the rule is embedded into the pre-state and the right-hand side into the post-state, and
- $o(L \setminus R) = G \setminus H$ and $o(R \setminus L) = H \setminus G$, i.e., precisely that part of $G$ is deleted which is matched by elements of $L$ not belonging to $R$ and, symmetrically, that part of $H$ is added which is matched by elements new in $R$.

A transformation sequence $G_0 \overset{p_1(o_1)}{\Longrightarrow} \cdots \overset{p_n(o_n)}{\Longrightarrow} G_n$ is a sequence of consecutive transformation steps.

On transformation sequences, a notion of equivalence is defined which generalises the permutation equivalence on firing sequences: two sequences are equivalent if they can be obtained from each other by repeatedly swapping independent transformation steps. This equivalence has been formalised by the notion of *shift-equivalence* [14] which is based on the following notion of independence of graph transformations. Two transformations $G \overset{p_1(o_1)}{\Longrightarrow} H_1 \overset{p_2(o_2)}{\Longrightarrow} X$ are *independent* if the occurrences $o_1(R_1)$ of the right-hand side of $p_1$ and $o_2(L_2)$ of the left-hand side of $p_2$ do only overlap in objects that are preserved by both steps, formally $o_1(R_1) \cap o_2(L_2) \subseteq o_1(L_1 \cap R_1) \cap o_2(L_2 \cap R_2)$. This is more sophisticated than the notion of independent firings of transitions which are required to use entirely disjoint resources.

### 4.2 Typed graph transformation with attributes

Assuming a set of data type symbols $S$, a *type graph with attribute declarations* (based on $S$) is a graph $TG$ whose set of vertices $TG_V$ contains $S$. Therefore, data type symbols are vertex types so that edges, representing attribute declarations, may be drawn towards them from ordinary vertices. This is compatible with notions of attributed graphs, like [17], where attribute carriers are used to relate graph elements and data values. Notice, however, that we limit ourselves to attributed vertices, and that we do not extend but refine the notion of graph.

Given a data domain $D_s$ for every data type symbol $s$, an *instance graph with attributes* over the type graph $TG$ is an instance graph $\langle G, g : G \to TG \rangle$ over $TG$ (in the above sense) such that $g^{-1}(s) = D_s \subseteq G_V$. Therefore, all vertices $x \in G_V$ with $g(x) \in S$ represent attribute values which may or may not be referenced by an edge from another vertex. As a consequence, instance graphs will be usually infinite. E.g., if the data type $\mathbb{N}$ of natural numbers is present, each $n \in \mathbb{N}$ will be a separate vertex.

Morphisms between instance graphs with attributes $\langle G, g : G \to TG \rangle$ and $\langle H, h : H \to TG \rangle$ are typed graph morphisms $f : G \to H$, i.e., graph morphisms compatible with the typing ($h \circ f = g$) and preserving the data domains; formally $f|_S = id_{G|_S}$, if we denote by $f|_S : G|_S \to H|_S$ the restriction of $f_V$ to vertices $x \in V$ of type $g(x) \in S$.

From this point on, all other notions, like rule, occurrence, transformation, transformation sequence, etc. are defined as in the previous subsection. Also, relevant results like the Local Church-Rosser Theorem, the Parallelism theorem, and the corresponding equivalence on transformation sequences based on shifting or swapping of independent transformations are easily transferred.

It is worth noticing that, in contrast with ER nets, attributes in our model are typed, that is, different types of nodes may have different selections of attributes. However, like in ER nets, our data types have no syntax: We only consider sets of values without explicit algebraic structure given by operations. As a consequence, we do not explicitly represent variables within rules and variable assignments as part of occurrences: A rule containing variables for attribute calculation and constraints is considered as a syntac-

tic representation of the (possibly infinite) set of its instances where the variables and expressions are replaced by concrete values.

### 4.3 Typed graph transformation with time

To incorporate time into typed graph transformation with attributes, we follow the approach of TER nets as discussed in Section 3. Therefore, a time data type is required as domain for time-valued attributes.

A *time data type* $T = \langle D_{time}, +, 0, \geq \rangle$ is a structure where $\geq$ is a partial order with $0$ as its least element. Moreover, $\langle +, 0 \rangle$ form a monoid (that is, $+$ is associative with neutral element $0$) and $+$ is monotone wrt. $\geq$. Obvious examples include natural or real numbers with the usual interpretation of the operations, but not dates in the YY:MM:DD format (e.g., due to the Y2K problem).

A *type graph with time* $TG$ is a type graph with attribute declarations based on a set of data type symbols $S$ that contains a special symbol $time$. An instance graph with time over $TG$ for a given time data type $T = \langle D_{time}, +, 0, \geq \rangle$ is an instance graph $\langle G, g : G \to TG \rangle$ over $TG$ such that the data type sort $time$ is interpreted by $D_{time}$, that is, $D_{time} = \{x \in G_V | g(x) = time\}$. Graph morphisms are defined as before.

Given a graph transformation rule $p : L \to R$ over a type graph with time, we say that

- *$p$ reads the chronos value $c$ of $v$* if $v \in L$ has a chronos attribute of value $c$, that is, there exists an edge $e \in L$ with $src(e) = v$ and $tar(e) = c \in D_{time}$.
- *$p$ writes the chronos value $c$ of $v$* if $v \in R$ has a chronos attribute of value $c$ which is not present in $L$, i.e., there exists an edge $e \in L$ with $src(e) = v$ and $tar(e) = c \in D_{time}$ and $e \notin L$.

Given a transformation $G \stackrel{p(o)}{\Longrightarrow} H$ we say that $p(o)$ reads/writes the chronos value of $w$ if there exists $v \in L \cup R$ such that $o(v) = w$ and $p$ reads/writes the chronos value of $v$.

It is important to note that, writing an attribute value of a vertex $v$ which is preserved by the rule (i.e., it belongs both to $L$ and $R$) means deleting the edge from $v$ to the old value and creating a new link to another value. Therefore, in this case, writing implies reading the value.

The definition of graph transformation rules with time has to take into account the particular properties of time as expressed, for example, by the axioms in Section 3. The direct transfer of axioms 1 and 2 leads to the following well-formedness conditions.

A *graph transformation rule with time* is a graph transformation rule over a type graph with time satisfying the following conditions.

**1. Local monotonicity:** All chronos values written by $p$ are higher than any of the chronos values read by $p$.
**2. Uniform duration:** All chronos values written by $p$ are equal.

A graph transformation system with time is an attributed graph transformation system over a type graph with time whose rules satisfy the conditions above.

This ensure a behaviour of time which can be described informally as follows. According to condition 1 an operation or transaction specified by a rule cannot take negative time, i.e., it cannot decrease the clock values of the nodes it is applied to. Condition

2 states an assumption about atomicity of rule application, that is, all effects specified in the right-hand side are observed at the same time. Given a transformation $G \overset{p(o)}{\Longrightarrow} H$ using rule $p$, this time *firing time*, is denoted by $time(p(o))$.

Notice that, due to the more general nature of typed graph transformation in comparison with ER nets, there exist some additional degrees of freedom.

**Existence of time-less vertex types:** ER nets are untyped (that is, all tokens have (potentially) the same attributes) while in typed graph transformation we can declare dedicated attributes for every vertex type. Therefore, we do not have to assume an attribute *chronos* for all vertex types, but could leave the decision about how to distribute chronos attributes to the designer. As we consider time as a distinguished semantic concept, which should not be confused with time-valued data, we do not allow more than one chronos attribute per vertex. This does not forbid us to model additional time-valued data by ordinary attributes.

**Update of chronos values for preserved vertices:** The second degree of freedom comes from the (well-known) fact that graph transformations generalise Petri nets by allowing contextual rewriting: All tokens in the post-domain of a transition are newly created while in the right-hand side of a graph transformation rule there may be vertices that are preserved. This allows to leave the chronos values of vertices in $L \cap R$ unchanged, creating new timestamps only for the newly generated items.

If we take in both cases the most restrictive choice, i.e., *chronos values for all types* and *update of chronos values for all vertices in $R$*, we can show, in analogy with TER nets, that for each transformation sequence $s$ using only rules that satisfy the above two conditions, there exists an equivalent sequence $s'$ such that $s'$ is time-ordered, that is, time is monotonically non-decreasing as the sequence advances.

**Theorem 1 (global monotonicity).** *Given a graph transformation system with time $\mathcal{G}$ such that*

- *its type graph declares a chronos attribute for every vertex type*
- *its rules write the chronos values of all vertices in their right-hand sides.*

*In this case, for every transformation sequence $s$ in $\mathcal{G}$ there exists an equivalent sequence $s' = G_0 \overset{p_1(o_1)}{\Longrightarrow} \ldots \overset{p_n(o_n)}{\Longrightarrow} G_n$ in $\mathcal{G}$ such that $s'$ is time-ordered, that is, $time(p_i(o_i)) \leq time(p_{i+1}(o_{i-1}))$ for all $i \in \{0, \ldots, n\}$.*

*Proof.* As a consequence of Theorem 2 below.

This is no longer true if we use the more liberal interpretation in any of the above choices, as shown by the following example.

*Example.* Figure 1 shows a type graph $TG$ and a (generic) instance graph $IG$ of $TG$, respectively. The type graph defines three vertex types: $T1, T2$ and $T3$. For $T2$ and $T3$ *chronos* attributes are declared, while $T1$ has no attribute. (Our example does not need edges.)

The instance graph contains nodes $A : T1$ (i.e., a $T1$-typed node named $A$), $B : T2$, and $C : T3$, where $C$ has the chronos value $c2$ and the chronos value of $B$ is $c2 + 3$.

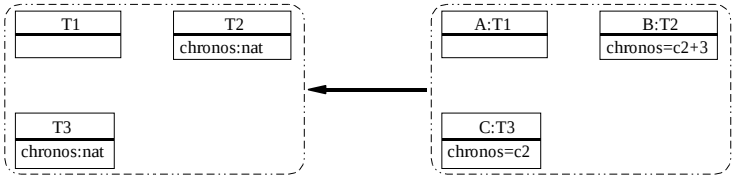


**Fig. 1.** Type graph (left) and instance graph (right)

Two rules, $p_1$ and $p_2$, are defined in Figure 2. By applying $p_1$, nodes $a : T1$ and $b : T2$ are matched and the chronos value of $b$ is increased by 4 time units.

Rule $p_2$ requires nodes $a : T1$ and $b : T3$. The former is deleted and the time of the latter is increased by 2 units. (Note, that the use of similar names does not imply any connection between the elements of different rules.)

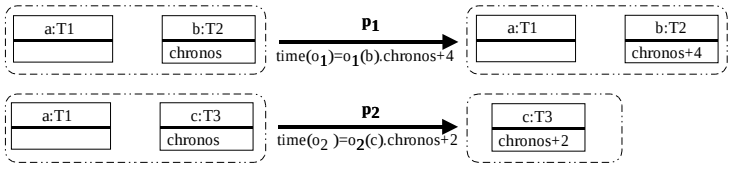


**Fig. 2.** Rules $p_1$ and $p_2$

An application of these rules to the instance graph in Figure 1 is shown in Figure 3. Both $p_1$ and $p_2$ are applicable to the graph. Applying first $p_1$ and then $p_2$ leads to the graph in the lower right. In this sequence, first the chronos value of $B$ is increased, and then $A$ is deleted and the chronos value of $C$ is increased. The occurrences and the firing times of the steps are denoted next to the arrows.

At this point, two observations are crucial. First, the two steps are not independent, that is there exists no equivalent sequence where $p_1$ and $p_2$ are applied in the reverse order. This is because $A \in (o_1(L_1 \cap R_1)) \cap o_2(L_2 \setminus R_2))$, i.e., $A$ is deleted by $p_2$ but required by $p_1$. Second, the sequence $IG \overset{p_1(o_1)}{\Longrightarrow} IG_1 \overset{p_2(o_2)}{\Longrightarrow} IG_2$ is not time-ordered because the firing time of the latter is smaller than the firing time of the first.

Note that, if $A$ would have a chronos attribute *and* all chronos values would be updated as required by Condition 2, $A$ should get $time(p_1(o_1))$ thus disabling its deletion at a lower time.

More generally, the problem is to ensure the consistency of causality and time in the sense that, whenever two steps are causally dependent, they must communicate their clock values. Indeed, this idea is crucial to many algorithms for establishing consistent

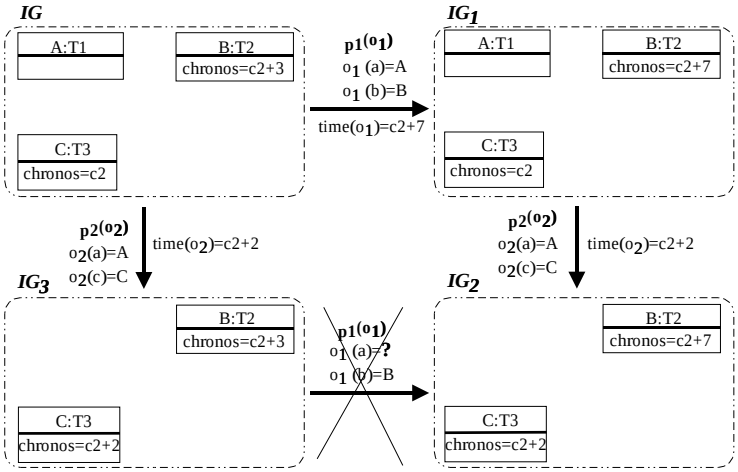


**Fig. 3.** Rule sequence

global time in distributed systems, based on logical clocks. The next theorem formalises this statement.

**Theorem 2 (global monotonicity).** *Given a graph transformation system with time $\mathcal{G}$ such that for all transformations $G \overset{p_1(o_1)}{\Longrightarrow} X \overset{p_2(o_2)}{\Longrightarrow} H$ in $\mathcal{G}$ that are* not *sequentially independent, there exists a vertex $v \in o_1(R_1) \cap o_2(L_2)$ whose chronos value is written by $p_1$ and read by $p_2$. In this case, for every transformation sequence $s$ in $\mathcal{G}$ there exists an equivalent sequence $s' = G_0 \overset{p_1(o_1)}{\Longrightarrow} \ldots \overset{p_n(o_n)}{\Longrightarrow} G_n$ in $\mathcal{G}$ such that $s'$ is time-ordered.*

*Proof.* The main line of the proof is as follows.

1. Our first observation is that the fact that two transformations $G \overset{p_1(o_1)}{\Longrightarrow} X \overset{p_2(o_2)}{\Longrightarrow} H$ are *not sequentially independent* implies that they are *time ordered*, i.e., $time(p_1(o_1)) \leq time(p_2(o_2))$. This is guaranteed by the existence of a common vertex $v \in o_1(R_1) \cap o_2(l_2)$ with a chronos value written by $p_1$ and read by $p_2$, which is
   (a) *exactly* the time of transformation $p_1(o_1)$ (due to the *"uniform duration"* condition),
   (b) *at most* the time of transformation $p_2(o_2)$ (as a consequence of the *"local monotonicity"* condition).
2. Then if two transformations are *not time ordered* and they are *sequentially independent*, we swap them in the rule application sequence[1] . We continue the swap operation until no such transformation pairs can be found.

[1] This algorithm is, in fact, conceptually similarly to the trick applied in the construction of a shift equivalent transformation sequence.

3. We state that after the *termination* of this swapping algorithm, a time ordered transformation sequence is obtained.
   (a) Let us suppose indirectly that there exist two transformations $G \overset{p_a(o_a)}{\Longrightarrow} X \overset{p_b(o_b)}{\Longrightarrow} H$ that violate the condition of time ordered sequences, i.e. $time(p_a(o_a)) > time(p_b(o_b))$.
   (b) However, if these transformations are *sequentially independent* then the algorithm in Item 2 can still be applied to them, which contradicts the assumption of termination.
   (c) On the other hand, if transformations $p_a(o_a)$ and $p_b(o_b)$ are *not sequentially independent* (but they are not time ordered by the indirect assumption), then we have a contradiction with our first observation, which established that two sequentially dependent transformations with a common vertex are always time ordered. □

Notice that the condition above can be effectively verified by checking all non-independent two-step sequences in $\mathcal{G}$ where $x = o_1(R_1) \cup o_2(L_2)$.

### 4.4 An example for time ordered sequences

In the sequel, our main theorem (Theorem 2) is discussed from a practical point of view on a small example of a communication system, which models *processes* sending *messages* to each other via *channels*. A message is sent via an *output* channel of a process, which *stores* the message until received via the *input* channel of the other process. The structure of our communication system is captured on the type graph of Fig. 4, while a sample system containing only two processes $P_1$ and $P_2$ with a single channel $C$ between them is also depicted there.

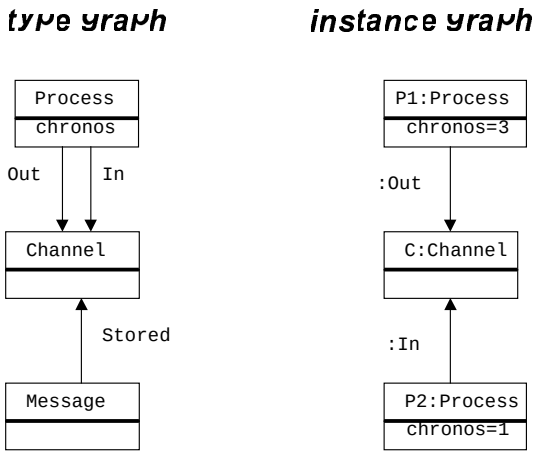


**Fig. 4.** A communication system: type and instance graphs

We introduce the following two basic operations (see Fig. 5) that can be performed in our system.

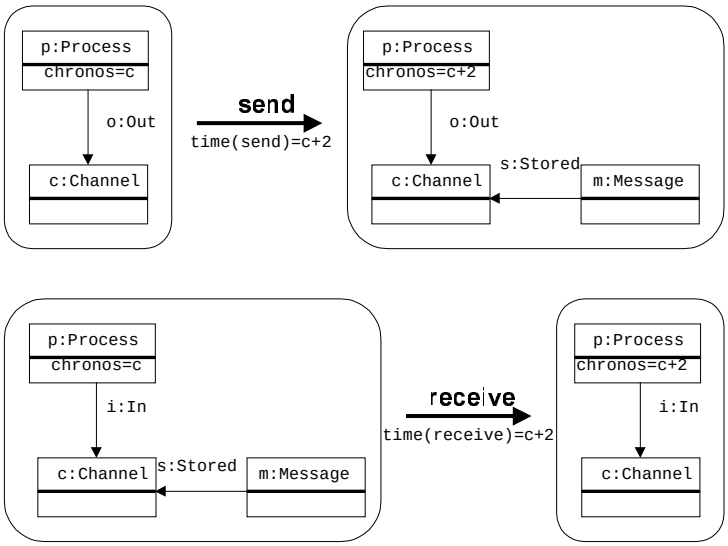


**Fig. 5.** Sending and receiving messages

- **Sending messages:** When process $p_1$ aims at sending a message, a message object $m$ is generated and placed into the output channel $c$. The application of the *send* rule takes 2 time units.
- **Receiving messages:** When a message $m$ arrives at the input port of a process $p$, then the process receives the message by removing the message object from the channel and destroying it afterwards. The application of *receive* rule takes 2 time units as well.

One can easily check that both rules satisfy the well-formedness conditions for graph transformation rules with time. However, the *send* rule computes its time from the chronos value of the sender process $P_1$, while the *receive* rule takes its time from (the chronos value of) the receiver process $P_2$, but no timestamps are attached to messages.

This turns out to be insufficient to guarantee the existence of time ordered transformation sequences, when considering, for instance, the transformation depicted in Fig. 6. In this case, the clock of the sender process $P_1$ is ahead of the chronos value of the receiver process ($chronos(P_1) = 3$ vs. $chronos(P_2) = 1$). Since no timestamps are attached when a message is sent, the receiver cannot synchronise its clock to the sender when the message is processed yielding a transformation sequence that is not time ordered ($time(send) = 5$ but $time(receive) = 3$ as each operation takes 2 time

units). However, we cannot swap the two transformations $send$ and $receive$ since they are not sequentially independent.

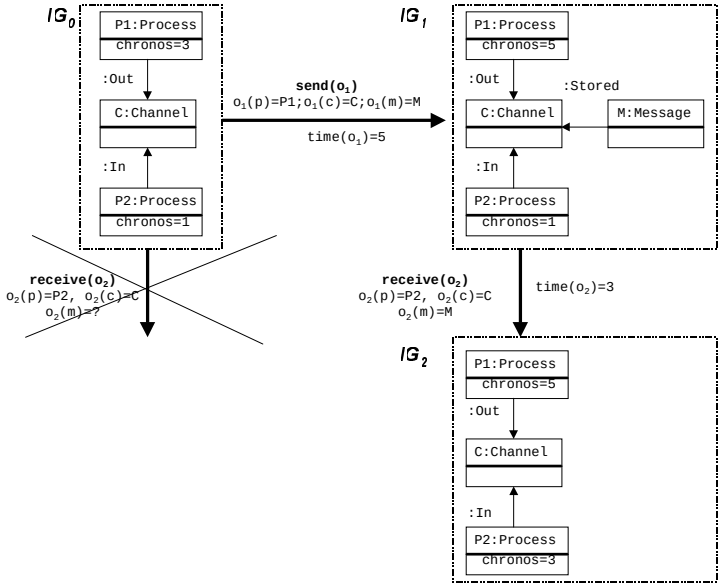


**Fig. 6.** A transformation violating time orderedness

It can be observed how the condition of Theorem 2 is violated. While (the application of) *send* is not sequentially independent from *receive* (a message object is created by *send* and required as a precondition by *receive*), there are no graph objects with chronos values written by *send* (as *send* rule only writes the chronos of process $P_1$) and read by *receive* (which only reads the chronos of process $P_2$).

The solution to the problem is well-known: a timestamp is needed to be attached to each message (see the corresponding rule in Fig 7). In terms of graph transformation systems with time, the *send* rule needs to write the chronos attribute of the message, while *receive* rule is required to read the chronos value of the message in order to synchronise its own clock.

This time, our global monotonicity theorem trivially holds, since the chronos value of each *message* object is written by the *send* rule and read by the *receive* rule. Thus in a transformation sequence where a certain application of *send* precedes the application of *receive*, the time of *receive* cannot be less then the time of *send* due to the well-formedness conditions 1 and 2.

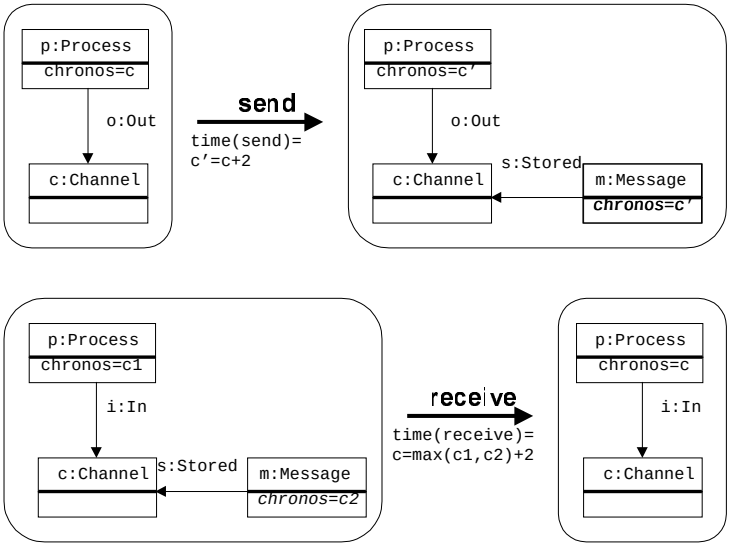


**Fig. 7.** Synchronisation by timestamps

## 5 Conclusion

We have transferred the model of time within ER nets, a kind of high-level Petri nets, to graph transformation systems. The resulting notion is a special case of typed graph transformation, where certain vertices are interpreted as time values and edges towards these vertices are time-valued attributes.

We have discussed some choices and their semantic consequences leading to the establishment of a *global monotonicity theorem*, which provides a sufficient condition for the existence of time ordered transformation sequences. This theorem generalises the idea behind familiar algorithms for establishing consistent logical clocks via time stamps in distributed systems [16], which are based on more specific computational models similar to the example in the last subsection.

It requires a deeper analysis of potential applications, in particular, the use of time in diagrammatic techniques like statecharts or sequence diagrams and their existing formalisations within graph transformation [7, 12, 13, 15], to understand if our choices are the right ones.

*Acknowledgement.* We wish to thank Luciano Baresi for his introduction to Petri nets with time.

## References

1. P. Baldan. *Modelling Concurrent Computations: from Contextual Petri Nets to Graph Grammars*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 2000.

2. P. Baldan, A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Concurrent semantics of algebraic graph transformation. In H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency and Distribution*, pages 107 – 188. World Scientific, 1999.
3. L. Baresi, M. Pezzé, and G. Taentzer, editors. *Proc. ICALP 2001 Workshop on Graph Transformation and Visual Modeling Techniques, Heraklion, Greece*, Electronic Notes in TCS. Elsevier Science, July 2001.
4. A. Corradini and R. Heckel, editors. *Proc. ICALP 2000 Workshop on Graph Transformation and Visual Modeling Techniques*, Geneva, Switzerland, July 2000. Carleton Scientific. `http://www.di.unipi.it/GT-VMT/`.
5. A. Corradini and U. Montanari. Specification of Concurrent Systems: from Petri Nets to Graph Grammars. In *Quality of Communication-Based Systems*, pages 35–52. Kluwer Academic Publishers, 1995.
6. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26(3,4):241–266, 1996.
7. G. Engels, J.H. Hausmann, R. Heckel, and St. Sauer. Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In A. Evans, S. Kent, and B. Selic, editors, *Proc. UML 2000, York, UK*, volume 1939 of *LNCS*, pages 323–337. Springer-Verlag, 2000.
8. H. Ehrig, J. Padberg, and L. Ribeiro. Algebraic high-level nets: Petri nets revisited. In *Recent Trends in Data Type Specification*, pages 188–206, Caldes de Malavella, Spain, 1994. Springer Verlag. Lecture Notes in Computer Science 785.
9. H. Ehrig, M. Pfender, and H.J. Schneider. Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973.
10. S. Gyapay and R. Heckel. Towards graph transformation with time. In *Proc. ETAPS'02 Workshop on Application of Graph Transformation, Grenoble, France*, 2002.
11. C. Ghezzi, D. Mandrioli, S. Morasca, and Pezzè. A unified high-level petri net formalism for time-critical systems. *IEEE Transactions on Software Engineering*, 17(2):160–172, 1991.
12. J.H. Hausmann, R. Heckel, and S. Sauer. Towards dynamic meta modeling of UML extensions: An extensible semantics for UML sequence diagrams. In M. Minas and A. Schürr, editors, *Symposium on Visual Languages and Formal Methods, IEEE Symposia on Human Computer Interaction (HCI 2001), Stresa, Italy*, Los Alamitos, CA, September 2001. IEEE Computer Society Press.
13. C. E. Hrischuk. A Model Making Automation Process (MMAP) using a graph grammar formalism. In *Theory and Application of Graph Transformations, 6th International Workshop, TAGT'98, Paderborn, Germany, Selected Papers*, volume 1764 of *LNCS*, pages 442–454. Springer-Verlag, 2000.
14. H.-J. Kreowski. *Manipulation von Graphmanipulationen.* PhD thesis, Technical University of Berlin, Dep. of Comp. Sci., 1977.
15. S. Kuske. A formal semantics of UML state machines based on structured graph transformation. In M. Gogolla and C. Kobryn, editors, *Proc. UML 2001, Toronto, Kanada*, volume 2185 of *LNCS*, pages 241–256. Springer-Verlag, 2001.
16. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7), July 1978.
17. M. Löwe, M. Korff, and A. Wagner. An algebraic framework for the transformation of attributed graphs. In M. R. Sleep, M. J. Plasmeijer, and M.C. van Eekelen, editors,

*Term Graph Rewriting: Theory and Practice*, chapter 14, pages 185–199. John Wiley & Sons Ltd, 1993.

18. L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars*. PhD thesis, TU Berlin, 1996.