Autonomous Units and their Semantics – the Sequential Case

Karsten Hölscher, Hans-Jörg Kreowski, and Sabine Kuske

University of Bremen, Department of Computer Science
P.O.Box 33 04 40, 28334 Bremen, Germany
{hoelscher,kreo,kuske}@informatik.uni-bremen.de

Published in:
A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, G. Rozenberg, editors, Proc. 3rd Intl. Conference on Graph Transformations (ICGT 2006), Volume 4178 of Lecture Notes in Computer Science. Springer, 2006.

Autonomous Units and their Semantics — the Sequential Case*

Karsten Hölscher, Hans-Jörg Kreowski, and Sabine Kuske

University of Bremen, Department of Computer Science
P.O.Box 330440, D-28334 Bremen, Germany
{hoelscher,kreo,kuske}@informatik.uni-bremen.de

Abstract. In this paper, we introduce the notion of a community of autonomous units as a rule-based and graph-transformational device to model processes that run interactively but independently of each other in a common environment. The emphasis of the approach is laid on the study of the formal semantics of a community as a whole and of each of its member units separately. We concentrate on the sequential case where only one unit can act at a time and the rule applications of the involved units are interleaved with each other.

1 Introduction

Data processing of today (like communication networks, multiagent systems, swarm intelligence, ubiquitous, wearable and mobile computing) is often distributed and comprises various components that run partially independent of each other, but may access and update the same information structures, communicate with each other and interact in various ways. They may cooperate to reach a common goal or may compete with each other to achieve their individual aims. Typical examples of this kind are logistic processes and systems like transport and production networks where many actors from different companies come together and cooperate to a certain degree. But they are usually still competitors who are not willing to transfer their control to others or to a central entity. On the more technical level, transport networks, for example, comprise many transport vehicles, lots of goods to be shipped, various further components for storing, loading, reloading, etc. It is not meaningful to model such a network as a centralized system with a single control. The same applies to production networks with respect to the involved machines, materials, storage areas, etc.

The main idea of this paper is to provide a formal graph-transformational and rule-based framework for the modeling of such systems composed of a variety of highly self-controlled components that make their decisions on their own depending on the information they get from their environment.

* Research partially supported by the EC Research Training Network SegraVis (Syntactic and Semantic Integration of Visual Modeling Techniques) and the Collaborative Research Centre 637 (Autonomous Cooperating Logistic Processes: A Paradigm Shift and Its Limitations) funded by the German Research Foundation (DFG).

The basic notion is that of a community of autonomous units which exist in a common environment (assumed to be a graph). There are initial environments to start computational processes, and there is an overall goal. Each autonomous unit in a community has its own individual goal in addition. To reach its goal, the unit can apply its rules or ask imported units for help. Moreover, each unit has a control mechanism to decide which rule is applied next or which imported unit is used next. This establishes the autonomy of a unit.

In this paper, we concentrate on the sequential semantics of communities of autonomous units. The semantics is given by all sequential processes - finite and infinite - that start in an initial environment, are composed of rule applications of autonomous units and calls of imported units, and follow, in each step, the control of the active unit. From the point of view of a single unit, this means that its own actions (being rule applications or calls of imported units) take place interleaved with other changes of the dynamic environment caused by the coexisting units. Clearly, the sequential semantics is only adequate if one deals with systems in which activities take place one after the other. Examples of this kind are card and board games, sequential algorithms, single-processor systems and such. Moreover, there are many modeling approaches the semantics of which assumes one action at a time. But even sequential systems may consist of self-controling components that decide about their own activities independently of the others like the examples of card and board games with several players show.

Autonomous units generalize our former modeling concept of graph transformation units (see, e.g., [1]). While the latter apply their rules and call imported units without any interference from the outside, an autonomous unit works in a dynamic environment which may change because of the activities of other units in the community. This makes a tremendous difference because the running of the system is no longer controlled by a central entity.

The benefit we expect of using autonomous units is to obtain an easy-to-use and visually well-understandable formal framework with a precise semantics that allows to model systems of interacting components so that on the one hand external control structures are set aside and on the other hand string-based representation is replaced by graph- and rule-based representation that allows to visualize and specify the systems more like they are. Nevertheless, the presented concepts are not restricted to graphs and graph transformation but can be used for any rule-based mechanism where rules modify some kind of configurations (cf. also [2]).

The paper is organized as follows. In Sect. 2 we briefly recall the notion of a graph transformation approach. In Sect. 3 autonomous units are introduced and a sequential semantics for them is given. Section 4 presents communities of autonomous units and how they interact within a common environment. Section 5 compares communities of autonomous units with the original transformation units introduced and studied in [1]. In Sect. 6 we present a case study modeling the players of the board game *Ludo* as autonomous units. The conclusion is given in Sect. 7. For reasons of space limitations proofs are omitted in this paper.

2 Graph Transformation Approaches

Whenever one has to do with dynamic graph-like structures, graph transformation (see also [3]) constitutes an adequate formal specification technique because it supports the visual and rule-based transformation of such structures in an intuitive and direct way. The ingredients of graph transformation are provided by a so-called graph transformation approach. In this section, we recall the notion of a graph transformation approach as introduced in [1] but modified with respect to the class of control conditions.

Two basic components of every graph transformation approach are a class of graphs, and a class of rules that can be applied to these graphs. In many cases, rule application is highly nondeterministic — a property that is not always desirable. Hence, graph transformation approaches can also provide a class of control conditions so that the degree of nondeterminism of rule application can be reduced. Moreover, graph class expressions can be used in order to specify for example sets of initial and terminal graphs of graph transformation processes.

Formally, a graph transformation approach is a system $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \mathcal{X}, \mathcal{C})$ the components of which are defined as follows.

- $\mathcal{G}$ is a class of *graphs*.
- $\mathcal{R}$ is a class of *graph transformation rules* such that every $r \in \mathcal{R}$ specifies a binary relation on graphs $SEM(r) \subseteq \mathcal{G} \times \mathcal{G}$.
- $\mathcal{X}$ is a class of *graph class expressions* such that each $x \in \mathcal{X}$ specifies a set of graphs $SEM(x) \subseteq \mathcal{G}$.
- $\mathcal{C}$ is a class of *control conditions* such that each $c \in \mathcal{C}$ specifies a set of sequences $SEM_{E,Change}(c) \subseteq SEQ(\mathcal{G})$ where $E\colon ID \to 2^{SEQ(\mathcal{G})}$, for some set ID of names and $Change \subseteq \mathcal{G} \times \mathcal{G}$.[1] As we will see later the mapping E is meant to associate a semantics to rules and imported autonomous units. The relation *Change* defines the changes that can occur in the environment of an autonomous unit. Hence, control conditions have a loose semantics which depends on the semantics associated to rules and imported units via the mapping E and on the changes of the environment given by *Change*.

For technical simplicity we assume in the following that ID is an arbitrary but fixed set with $\mathcal{R} \subseteq ID$ and that $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \mathcal{X}, \mathcal{C})$ is an arbitrary but fixed graph transformation approach.

3 Autonomous Units

Autonomous units act within or interact on a common environment which is modeled as a graph. An autonomous unit consists of a set of graph transformation rules, a control condition, and a goal. Moreover, it can import other units to which it may delegate auxiliary tasks. The graph transformation rules contained

[1] For a set A 2^A denotes its powerset and $SEQ(A)$ the set of finite and infinite sequences over A.

in an autonomous unit *aut* and the imported units of *aut* specify all transformations the unit *aut* can perform. Such a transformation comprises for example a movement of the autonomous unit within the current environment, the exchange of information with other units via the environment, or local changes of the environment. The control condition regulates the application process. For example, it may require that a sequence of rules be applied as long as possible or infinitely often. In this first approach the goal of a unit is a graph class expression determining how the transformed graphs should look like.

Definition 1 (Autonomous unit). An *autonomous unit* is a system $aut = (g, U, P, c)$ where $g \in \mathcal{X}$ is the *goal*, U is a set of imported autonomous units, $P \subseteq \mathcal{R}$ is a set of graph transformation rules, and $c \in \mathcal{C}$ is a control condition. The components of *aut* are also denoted by g_{aut}, U_{aut}, P_{aut}, and c_{aut}, respectively.

In the following we consider only autonomous units with acyclic import structure. Moreover, for technical simplicity we assume that in addition to the rules all autonomous units are contained in the set ID.

An autonomous unit modifies an underlying environment while striving for its goal. Its semantics consists of a set of transformation processes being finite or infinite sequences of environment transformations. An environment transformation comprises the application of a local rule or a transformation process performed by an imported unit or an environment change typically performed by another autonomous unit that is working in the same environment. These environment changes are given as a binary relation of environments. Hence, in this sequential approach a transformation process of an autonomous unit interleaves local rule applications with transformation processes of imported units and environment changes specified by other components. This implies that an environment transformation of an imported unit cannot be interrupted by changes of the importing unit but it can be interleaved with the change relation induced by other components. Hence, every autonomous unit has exactly one thread of control. Autonomous units regulate their transformation processes by choosing in every step only those rules or imported units that are allowed by its control condition.

The definition of the sequential semantics of autonomous units makes use of the sequential composition of sequences. Let $s = (x_0, \dots, x_n)$ and $s' = (x'_0, x'_1, \dots)$ be sequences such that s is finite, s' is finite or infinite, and $x'_0 = x_n$. Then the *sequential composition* of s and s' is equal to $s \circ s' = (x_0, \dots x_n, x'_1, \dots)$. Moreover, the number of elements of a finite sequence $s = (x_0, \dots, x_n)$ is equal to $n + 1$ and is denoted by $|s|$. For an infinite sequence s its number of elements is $|s| = \infty$. The first element of a sequence s is denoted by $\mathit{first}(s)$ and its last element by $\mathit{last}(s)$ in the case where s is finite.

Definition 2 (Sequential semantics). Let $aut = (g, U, P, c)$ be an autonomous unit and let $\mathit{Change} \subseteq \mathcal{G} \times \mathcal{G}$. Let $s \in \mathit{SEQ}(\mathcal{G})$. Then $s \in \mathit{SEM}_{\mathit{Change}}(aut)$ if

- there is a sequence $seq = (s_0, s_1, \dots) \in \mathit{SEQ}(\mathit{SEQ}(\mathcal{G}))$ such that
  - for $0 \leq i < |seq|$ if seq is finite and for all $i \in \mathbb{N}$ if seq is infinite, s_i is finite and $\mathit{last}(s_i) = \mathit{first}(s_{i+1})$;

- $s = s_0 \circ s_1 \circ \cdots$; [2]
- for $i = 0, \ldots, |seq|$ if seq is finite and for all $i \in \mathbb{N}$ if seq is infinite

$$s_i \in \bigcup_{p \in P} SEM(p) \cup Change \cup \bigcup_{u \in U} SEM_{Change}(u).$$

– $s \in SEM_{E(aut),Change}(c)$ with $E(aut)(id) = SEM(id)$ if $id \in P$, $E(aut)(id) = SEM_{Change}(id)$ if $id \in U$, and $E(aut)(id) = \emptyset$, otherwise.

It is worth noting that the semantics of autonomous units is inductively defined meaning that it covers the case where no unit is imported and in the case where the set of imported units is not empty the semantics of every imported unit is recursively computed.

Every autonomous unit induces a set of atomic (i.e. not interruptible) environment transformations that consist of the semantic relation of all local rules plus the atomic transformations of the imported autonomous units.

Definition 3 (Atomic transformations). The set of *atomic transformations* of an autonomous transformation unit $aut = (g, U, P, c)$ is defined as $AT(aut) = \bigcup_{p \in P} SEM(p) \cup \bigcup_{u \in U} AT(u)$.

As one would expect a transformation process of an autonomous unit consists of a sequence of atomic transformations interleaved with other changes of the environment.

Observation 1 Let $aut = (g, U, P, c)$ be an autonomous unit and let $s = (G_0, G_1, \ldots) \in SEM_{Change}(aut)$ for some relation $Change \subseteq \mathcal{G} \times \mathcal{G}$. Then for $i = 1, \ldots, |s|$ if s is finite and for all $i \in \mathbb{N}^+$ if s is infinite $(G_{i-1}, G_i) \in AT(aut) \cup Change$.[3]

4 Communities of Autonomous Units

Autonomous units are meant to work within a community of autonomous units that modify the common environment together. In the sequential case these modifications take place in an interleaving manner. Every community is composed of an overall goal that should be achieved, an environment specification that specifies the set of initial environments the community may start working with, and a set of autonomous units. The overall goal may be closely related to the goals of the autonomous units in the community. Typical examples are the goals admitting only graphs that satisfy the goals of one or all autonomous units in the community.

Definition 4 (Community). A *community* is a triple $COM = (Goal, Init, Aut)$, where $Goal, Init \in \mathcal{X}$ are graph class expressions called the *overall goal* and the *initial environment specification*, respectively, and Aut is a set of autonomous units.

[2] Please note that $s = s_0 \circ s_1 \circ \cdots$ stands for $s = s_0 \circ s_1 \circ \cdots \circ s_{|seq|}$ if seq is finite.

[3] $\mathbb{N}^+$ denotes the set of all positive natural numbers.

In a community all units work on the common environment in a self-controlled way by applying their rules or their imported units. The change relation integrated in the semantics of autonomous units makes it possible to define an interleaving semantics of a community in which every autonomous unit may perform its transformation processes. For this purpose it is necessary to know for every autonomous unit the set of atomic transformations of all other units in the community.

Definition 5 (Change relation). Let $COM = (Goal, Init, Aut)$ be a community. Then for each $aut \in Aut$ the *change relation* w.r.t. aut is defined as $Change(aut) = \bigcup_{aut' \in Aut - \{aut\}} AT(aut')$.

Every transformation process of a community must start with a graph specified as an initial environment of the community. Moreover, it must be in the sequential semantics of every autonomous unit participating in the community. A finite transformation process of a community is successful if its last environment satisfies the overall goal. Every infinite transformation process of a community is successful if it meets infinitely many environments that satisfy the overall goal.

Definition 6 (Sequential community semantics).

1. Let $COM = (Goal, Init, Aut)$. Then the *sequential community semantics* of COM consists of all finite or infinite sequences $s = (G_0, G_1, \ldots) \in SEQ(\mathcal{G})$ such that $G_0 \in SEM(Init)$ and $s \in SEM_{Change(aut)}(aut)$ for all $aut \in Aut$.
2. The sequence s is called a *successful transformation process* if s is finite and $G_{|s|} \in SEM(Goal)$ or if for all $j \in \mathbb{N}$ there is a finite sequence $s_j = (G_{j,0}, \ldots, G_{j,n_j})$ with $G_{j,n_j} \in SEM(Goal)$ such that $s = s_0 \circ s_1 \circ \cdots$. The set of all successful transformation processes of COM is denoted by $STP(COM)$.

As the definition of the community semantics shows, there is a strong connection between the semantics of a community $COM = (Goal, Init, Aut)$ and the semantics of an autonomous unit $aut \in Aut$. More precisely, the semantics of COM is a subset of the semantics of aut w.r.t. the change relation $Change(aut)$.

For the community semantics we can also show in a straightforward way that only atomic transformations of the participating units are applied in every transformation process.

Observation 2 Let $COM = (Goal, Init, Aut)$ be a community and let $s = (G_0, G_1, \ldots) \in SEM(COM)$. Then for $i = 1, \ldots, |s|$ if s is finite and for all $i \in \mathbb{N}^+$ if s is infinite $(G_{i-1}, G_i) \in \bigcup_{aut \in Aut} AT(aut)$.

5 Comparison with Transformation Units

In this section we compare communities of autonomous units with transformation units that have an acyclic import structure (see e.g. [1]). Autonomous units are up to a certain degree similar to transformation units because both concepts are graph- and rule-based, use control conditions, import other units, and employ graph class expressions. Nevertheless there are some fundamental differences: (1) While an autonomous unit interacts with other autonomous units, an ordinary transformation unit runs its computations without interference of other units except those imported. Hence the semantics of transformation units is not defined with respect to possible environment changes. (2) All transformations are solely controlled by the transformation units whereas an autonomous unit controls its own actions, but not those of the other units in the community. (3) The semantics of a transformation unit and consequently also the semantics of control conditions used in transformation units are binary relations on graphs whereas the semantics of autonomous units consists of finite and infinite transformation processes, i.e. finite and infinite sequences of graphs. Because of these differences communities have the advantage that systems such as logistic processes or games consisting of many automomously and perhaps infinitely long acting components can be modeled in a more realistic way.

If one considers only the semantic relation induced by all finite transformation sequences of communities and if there exist appropriate control conditions in the underlying graph transformation approach, one can translate transformation units into communities. To this aim we define for every finite sequence $s = (G_0, \ldots, G_n)$ its *induced pair* as $pair(s) = (G_0, G_n)$ and for every set S of sequences its *induced binary relation* as $rel(S) = \{pair(s) \mid s \in S'\}$ where S' is the set of all finite sequences in S. Moreover, for a set $\mathcal{G}$, its identity relation is the set $\Delta\mathcal{G} = \{(G, G) \mid G \in \mathcal{G}\}$. Finally, for a binary relation $R \subseteq \mathcal{G} \times \mathcal{G}$ the set R^* denotes the set of all finite sequences obtained from sequentially composing pairs of $R \cup \Delta\mathcal{G}$, i.e. $R^* = \{(r_0, \ldots, r_n) \mid (r_{i-1}, r_i) \in R \cup \Delta\mathcal{G}$ for $i = 1, \ldots, n, n \geq 1\}$.[4]

5.1 Transformation Units

A *transformation unit* is a system $tu = (I, U, P, C, T)$ where $I, T \in \mathcal{X}$, U is a set of imported transformation units, $P \subseteq \mathcal{R}$, and C is a control condition that specifies for every mapping $E\colon ID \to 2^{\mathcal{G}\times\mathcal{G}}$ a binary relation on graphs. Please note that analogously to autonomous units, transformation units are also inductively defined i.e. they have an acyclic import structure.[5] The set of *directly and indirectly imported transformation units* of tu is inductively defined by $IMP(tu) = U \cup \bigcup_{u' \in U} IMP(u')$. A pair $(G, G') \in SEM(I) \times SEM(T)$ is in the *interleaving semantics* $SEM(tu)$ of tu if there is a sequence $s = (G_0, \ldots, G_n)$ of graphs such that $G_0 = G$, $G_n = G'$, for $i = 1, \ldots n$ $(G_{i-1}, G_i) \in \bigcup_{p \in P} SEM(p) \cup \bigcup_{u \in U} SEM(u)$, and $(G, G') \in SEM_{E(tu)}(C)$ where $E(tu)(id) = SEM(id)$ if $id \in P \cup U$ and $E(tu)(id) = \emptyset$, otherwise. Analogously to autonomous units the set of *atomic transformations* of tu is defined by $AT(tu) = \bigcup_{p \in P} SEM(p) \cup \bigcup_{u \in U} AT(u)$.

[4] Obviously, $rel(R^*)$ corresponds to the reflexive and transitive closure of R.

[5] Transformation units with an arbitrary import structure are studied in [4].

5.2 Translating Transformation Units into Communities

For comparing transformation units with communities we first translate every transformation unit tu into two sets of autonomous units namely $\mathit{TRANS1}(tu)$ and $\mathit{TRANS2}(tu)$. The units in both sets are inductively defined such that the rule set of every autonomous unit in $\mathit{TRANS1}(tu) \cup \mathit{TRANS2}(tu)$ is equal to the rule set of tu, the goal is equal to the terminal graph class expression of tu, and the control condition can be anyone satisfying a certain property that is different for $\mathit{TRANS1}(tu)$ and $\mathit{TRANS2}(tu)$.

Definition 7 (Translations). Let $tu = (I, U, P, C, T)$ be a transformation unit.

1. The *first translation* of tu is the set $\mathit{TRANS1}(tu)$ consisting of all autonomous units $aut(tu) = (T, \{aut(u) \mid u \in U\}, P, c)$ such that for all $s \in AT(tu)^*$ with $pair(s) \in \mathit{SEM}_{E(tu)}(C)$, $s \in \mathit{SEM}_{E(aut(tu)),\Delta\mathcal{G}}(c)$, and $aut(u) \in \mathit{TRANS1}(u)$ for each $u \in U$.
2. The *second translation* of tu is the set $\mathit{TRANS2}(tu)$ consisting of all autonomous units $aut(tu) = (T, \{aut(u) \mid u \in U\}, P, c)$ such that

$$rel(\mathit{SEM}_{E(aut(tu)),\Delta\mathcal{G}}(c)) \subseteq \mathit{SEM}_{E(tu)}(C),$$

   and $aut(u) \in \mathit{TRANS2}(u)$ for each $u \in U$.

It can be shown that the first translation preserves the behaviour of the original transformation unit tu (but can do more), and the second only performs such transformations that can also be done by tu, provided that the graph class expressions occurring in tu do not restrict the class $\mathcal{G}$ to some proper subclass. As a consequence we get that every autonomous unit in the intersection of $\mathit{TRANS1}(tu)$ and $\mathit{TRANS2}(tu)$ behaves as tu, if tu satisfies the mentioned property with respect to the graph class expressions.

These facts imply the following observation that relates transformation units with communities. In particular, for every autonomous unit aut in $\mathit{TRANS1}(tu) \cup \mathit{TRANS2}(tu)$ let $\mathit{COM}(aut)$ be the community with aut as its only autonomous unit, the initial graph class expression of tu as the initial environment specification, and the terminal expression of tu as the goal. Then the interleaving semantics of tu is contained in the binary relation induced by the transformation processes of $\mathit{COM}(aut)$ if aut belongs to the first translation of tu. Moreover, the binary relation induced by the successful transformation processes of $\mathit{COM}(aut)$ are contained in the interleaving semantics of tu if aut belongs to the second translation of tu. Consequently, the interleaving semantics of tu is equal to the binary relation induced by the successful transformation processes of COM if aut belongs to both translations. In the last two cases, the graph class expressions of all directly and indirectly imported transformation units of tu must specify the class of all graphs. It is worth noting that this condition concerning the graph class expressions can be dropped by requiring additionally that the control condition of every $aut(tu)$ in $\mathit{TRANS2}(tu)$ admits only sequences from initial into terminal graphs of tu.

Observation 3 Let $tu = (I, U, R, C, T)$ be a transformation unit and let $COM = (T, I, \{aut\})$. Then the following holds.

1. $SEM(tu) \subseteq rel(SEM(COM))$ if $aut \in TRANS1(tu)$.
2. $rel(STP(COM)) \subseteq SEM(tu)$ if $aut \in TRANS2(tu)$ and if $SEM(I_u) = SEM(T_u) = \mathcal{G}$ for all $u \in IMP(tu)$.
3. $SEM(tu) = rel(STP(COM))$ if $aut \in TRANS1(tu) \cap TRANS2(tu)$ and if $SEM(I_u) = SEM(T_u) = \mathcal{G}$ for all $u \in IMP(tu)$.

6 Modeling Ludo Players as Autonomous Units

Board games are a typical example of communities of autonomous units with sequential semantics where the board provides the common environment and the players are the autonomous units. As a concrete example we consider in this section the game *Ludo*. [6]

The graph transformation approach used in this example consists of labeled directed graphs and double-pushout rules (cf. [5]). The control conditions used are regular expressions and priorities. As graph class expressions we use subgraph conditions and the graph class expression specifying the class of all graphs. A *subgraph condition* is a graph G that admits all graphs that have (an isomorphic copy of) G as subgraph. Please note that in order to verify the presented case study we have implemented it based on the AGG system [5].

A possible environment graph of *Ludo* is the initial game situation where four players of different colours have all their tokens at the start place and there is one die showing an arbitrary number between one and six. This graph is depicted in Fig. 1. Every player is drawn as a kind of actor labeled with a colour out of b(lue), y(ellow), r(ed), and g(reen) so that every player has a different colour. Technically, a player is a labeled node. The players are connected via some directed edges indicating the playing direction. The game board consists of a start node and four home nodes for every player and a set of round nodes. The start node of a c-labeled player is depicted as a c-labeled polygon with six corners. The home nodes are drawn as rhombuses. Every c-labeled player has four c-labeled tokens that are all situated at her/his start node at the beginning of a match. The fact that a token of colour c is situated at a node v is visualized with a c-labeled token that is connected to v via an undirected edge. Technically, this can be modeled by means of a c-labeled loop connected to the node v. The directed edges between the nodes of the game board indicate where and in which direction the tokens can move around the game board.

Every round node and every directed edge between round and home nodes are labeled with a set $M \subseteq \{b, y, g, r\}$. The label of every round node contains all colours that can visit this node. Since at the beginning of a game all round nodes are vacant, i.e. they can be visited by all colours, they are all labeled with $\{b, y, g, r\}$. The labels of the edges connecting home and round nodes contain

[6] There exist several distinct versions of the game *Ludo*. In this paper we consider one of the standard german versions.

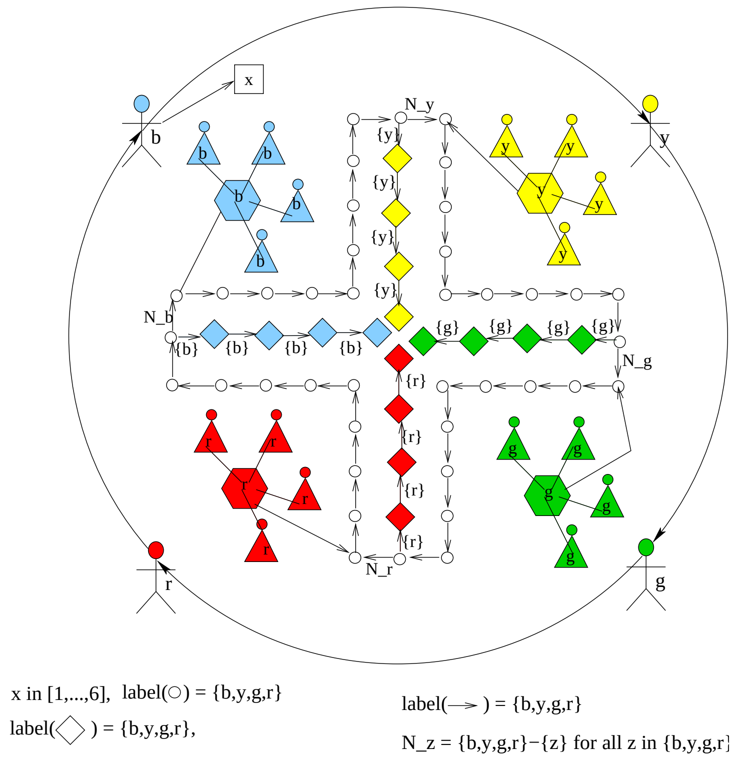
x
N_y
N_b
N_g
N_r
{y}
{b}
{g}
{r}
b
y
g
r
x in [1,...,6], label(○) = {b,y,g,r}
label(◇) = {b,y,g,r},
label(⟶) = {b,y,g,r}
N_z = {b,y,g,r}−{z} for all z in {b,y,g,r}

Fig. 1. An environment of *Ludo*

also all colours the tokens of which can move via these edges. For example, only yellow tokens can move to a home node of a yellow player. Moreover, no yellow token may go over the edge labeled with $N\_y = \{b, g, r\}$, because it has to enter its home. Please note that the labels of most of the nodes and edges of Fig. 1 are depicted below the graph in order to keep the graph easy to read.

The goal of every player is to have all four tokens at home, one in each home node. To reach a home place, a token must go from the start place over the round fields in the indicated direction. To move a token, a die must be thrown. If a six is thrown the current player must move one of her/his tokens from the start node to the first round node, i.e. to the round node connected to the start node. If there is no token left at the start node, the player can take any other of her/his tokens. A six allows for throwing again. We assume here that the blue player starts to play. This is why the *b*-labeled player is holding the die (represented by the edge from the player to the die). Afterwards it is the turn of the yellow player.

Every player of *Ludo* can be realized as the autonomous unit depicted in Fig. 2. The goal of a player is to have all of her/his tokens at home, one at each home node. The rules and the imported units model all possible actions

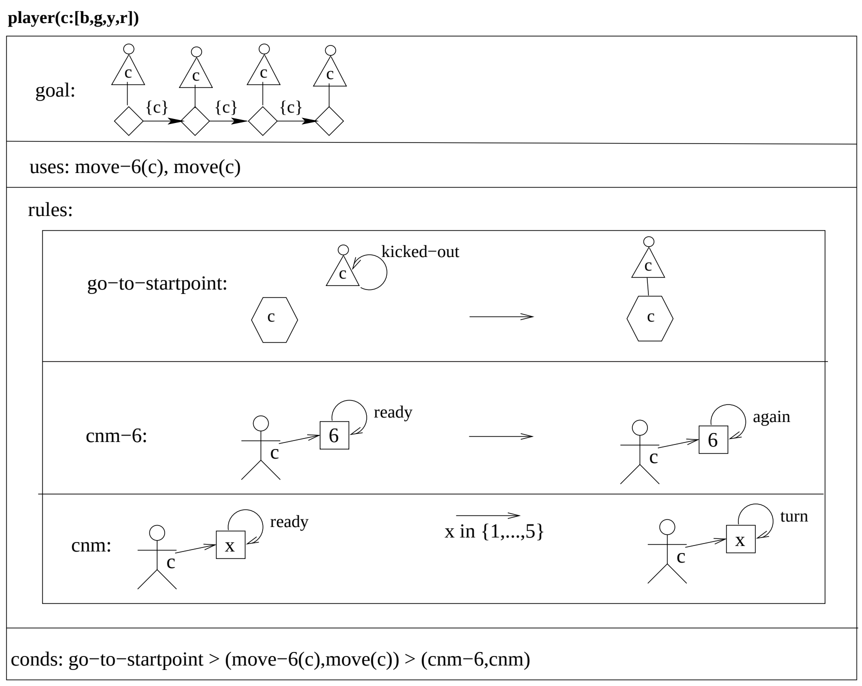

Fig. 2. A ludo player

of a player. A rule is depicted by an arrow pointing from the left-hand side of the rule to the right-hand side. The possible values of the variables occurring in the left- or right-hand side are put under the arrow. If the label of a node or an edge is not significant it is omitted in the rule, i.e. an item without a label can be matched to an item with any label. The rule *go-to-startpoint* of the unit *player* moves a token that has been kicked out to its home node. As the control condition prescribes this rule has the highest priority, i.e. it should be always applied if possible. The rule *cnm-6* and the rule set *cnm* are applied if no token can be moved by the player, i.e. they have the lowest priority. The rule *cnm-6* asks the die to throw again and *cnm* asks the die to turn to the next player if a number between one and five was thrown.

Every player imports the two units *move-6* and *move*. The unit *move* is depicted in Fig. 3. It models all moves of a token if no six is thrown. The moves corresponding to a six are contained in the unit *move-6*. For reasons of space limitations it is not depicted. The unit *move* contains four rules. The first, *mf* moves a token from the first round node (the one connected to its start node) x nodes ahead where $x \in \{1, \ldots, 5\}$ is the number thrown by the die. This move

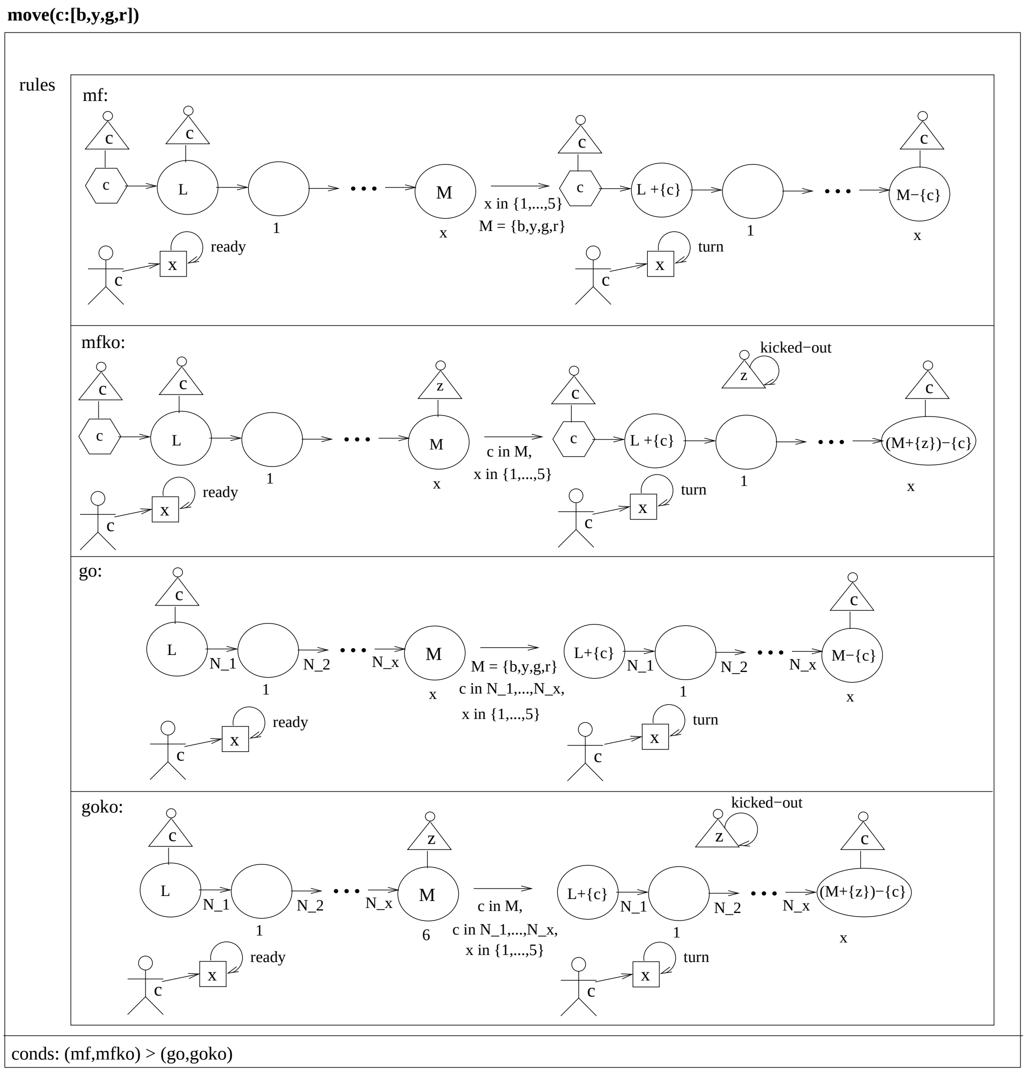

Fig. 3. The unit *move*

can only be performed if the target node is not occupied and if there is still a token at the start node. Moreover, the token can only be moved if it is the turn of its player. This is indicated by the arrow pointing from player *c* to the die. On the left-hand side the die has a *ready*-loop which means that the die has already thrown itself. On the right-hand side the die is asked to turn. The rule *mfko* is similar to *mf.* The difference is that another token is kicked out. The rule *go* moves from a round or a home node to another round or home node. The rule *goko* does the same but it additionally kicks out another token. The rules *go* and *goko* can only be applied if the first two rules are not applicable. This is why the first round node must be left if it is occupied by a token of colour *c* and if there is still a token at the start node. If this move is not possible any other token can be taken.

Please note that players select their tokens nondeterministically. More sophisticated rules would allow to decide whether it is appropriate to choose a token that can kick out another one, etc. The rules for making such decisions possible are more complicated, because they have to consider a wider context of the environment (e.g. such a rule could check whether the kicking out of another token brings the own token into a "dangerous" position). For reasons of space limitations they are kept simple in this paper.

The last autonomous unit of *Ludo* models the die and is depicted in Fig. 4. The unit *die* has no special goal, i.e. it admits every graph as a goal. The only

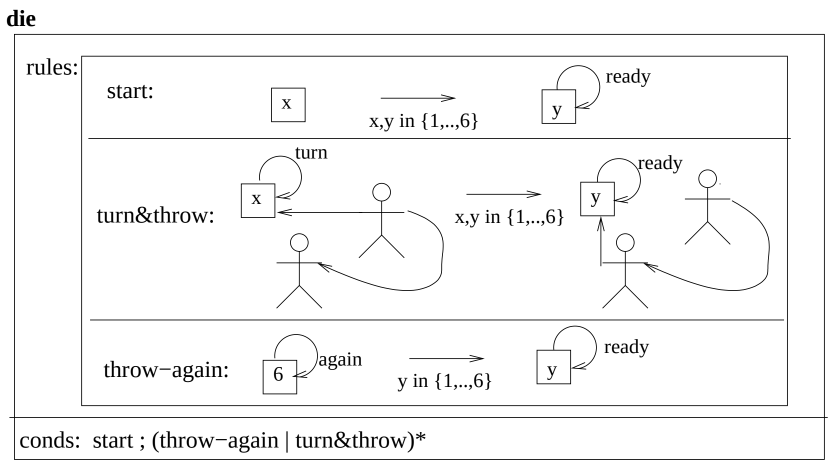

Fig. 4. The unit *die*

functionality of *die* is to throw itself and to move to the next player. The first rule throws the die. The second rule throws and turns the die in the case where the die gets a corresponding *turn*-message from the player. With the third rule the die throws itself again without moving to the next player. This can be only done if a six was thrown before. The control condition requires that the *start* rule be applied once at first. Afterwards any of the two remaining rules can be applied arbitrarily often.

The game *Ludo* (including the board, up to four players, one die and the game rules) can be modeled as the community *Ludo* the goal of which specifies all graphs in which at least one player unit has reached its goal, the initial environment specification specifies all possible start situations of *Ludo*, and the set autonomous units consists of the unit *die* and one player unit for every colour.

7 Conclusion

In this paper, we have introduced communities of autonomous units as a means for modeling systems in which different components interact in a rule-based, self-controlled, and goal-driven manner within a common environment. We have compared communities of autonomous units with the original transformation units and we have illustrated the notion of communities with a case study modeling the board game *Ludo* in which every player as well as the die can act as an autonomous unit.

The underlying formal framework for communities of autonomous units has been graph transformation which is highly adequate if the common environment can be represented in a natural way as a graph as for example in the case of board games and logistic applications. Nevertheless, it is worth noting that the graphs and the graph transformation rules the units are working with are not further specified in the underlying graph transformation approach so that in general, one can take as formal basis any rule-based mechanism that provides a set of configurations and a set of rules specifying a binary relation on such configurations.

The presented paper has been concentrated on the sequential semantics of communities of autonomous units which seems to be suitable for a series of applications. For applications in which unit actions can happen in parallel or concurrently, the semantics can be defined accordingly as is done in a very first attempt in [6] where one can find also a case study from the area of transport logistics.

There are at least the following interesting points for future work. (1) Communities of autonomous units should be compared with related approaches such as agent systems [7, 8], swarm intelligence [9] and distributed graph transformation [10]. (2) Communities of autonomous units should be implemented in order to be able to elaborate and to verify case-studies of realistic size. In order to verify the case study presented in Sect. 6 we have implemented it based on the AGG system. For this pupose the community *Ludo* had to be translated into a single flat graph transformation system because AGG does not support the concept of transformation units or communities. Currently there is being done some work towards the implementation of communities at the University of Bremen which has as one aim to allow to plug in other already existing graph transformation tools (cf. [11]). (3) Up to now, the goal of an autonomous unit is defined as a graph class expression. Since for some applications this may not be sufficient, other adequate classes of goals should be studied. It should also be studied which concrete classes of control conditions are the right ones for autonomous units. In particular, in the case of games different playing strategies should be employed for the different units in a communities. (4) More case studies should be specified. In this context it would be meaningful to elaborate for one application several case studies with a different degree of control distribution. For example in the case of the game *Ludo* one could compare the presented case study with another in which also the tokens are modeled as autonomous units with their own control and rules. (5) Communities of autonomous units that can perform parallel and concurrent transformations should be further investigated taking into account concepts and results from concurrent, parallel and distributed graph transformation (cf.,e.g., [12, 10]).

References

1. Kreowski, H.J., Kuske, S.: Graph transformation units with interleaving semantics. Formal Aspects of Computing **11**(6) (1999) 690–723
2. Kreowski, H.J., Kuske, S.: Approach-independent structuring concepts for rule-based systems. In Wirsing, M., Pattison, D., Hennicker, R., eds.: Proc. 16th Int. Workshop on Algebraic Development Techniques (WADT 2002). Volume 2755 of Lecture Notes in Computer Science. (2003) 299–311
3. Rozenberg, G., ed.: Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations. World Scientific, Singapore (1997)
4. Kreowski, H.J., Kuske, S., Schürr, A.: Nested graph transformation units. International Journal on Software Engineering and Knowledge Engineering **7**(4) (1997) 479–502
5. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G., eds.: Fundamentals of Algebraic Graph Transformation. Springer (2006)
6. Hölscher, K., Klempien-Hinrichs, R., Knirsch, P., Kreowski, H.J., Kuske, S.: Regelbasierte Modellierung mit autonomen Transformationseinheiten. Technical Report 1, University of Bremen (2006)
7. Weiss, G., ed.: Multiagent Systems — A Modern Approach to Distributed Artificial Intelligence. The MIT Press (1999)
8. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. The Knowledge Engineering Review **10**(2) (1995)
9. Kennedy, J., Eberhart, R.C.: Swarm Intelligence. Morgan Kaufmann (2001)
10. Ehrig, H., Kreowski, H.J., Montanari, U., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3: Concurrency, Parallelism, and Distribution. World Scientific, Singapore (1999)
11. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools. World Scientific, Singapore (1999)
12. Heckel, R.: Open Graph Transformation Systems: A New Approach to the Compositional Modelling of Concurrent and Reactive Systems. PhD thesis, Technical University of Berlin (1998)