Published in: Hülsmann, M.; Windt, K. (eds.): Understanding Autonomous Cooperation & Control in Logistics – The Impact on Management, Information and Communication and Material Flow. Springer, Berlin, 2007

Autonomous Units: Basic Concepts and Semantic Foundation

Karsten Hölscher[1], Renate Klempien-Hinrichs[2], Peter Knirsch[1], Hans-Jörg Kreowski[1], Sabine Kuske[1]

[1]Faculty for Mathematics and Computer Science, University of Bremen, Bremen, Germany
[2]Faculty for Production Engineering, University of Bremen, Bremen, Germany

Introduction

Today, most data processing systems and most logistic systems comprise various, possibly distributed, components. These components typically act autonomously, but they may also communicate and interact with each other, spontaneously linking up to form a network. These components do not necessarily need to be stationary. Sometimes they even move or are carried around. Although the components act autonomously, the task to be solved is handled by their interaction and the system as a whole. In this paper the concept of autonomous units for modeling such systems is proposed. Autonomous units form a community with a common environment, in which they act and which they transform. Autonomous units are based on rules, the applications of which yield changes in the environment. They are also equipped with an individual goal, which they try to accomplish by applying their rules. A control condition enables autonomous units to select at any time and in any situation the rule that should actually be applied from the set of all applicable rules.

The motivation for introducing autonomous units as a modeling concept arises from the Collaborative Research Centre 637 *Autonomous Cooperating Logistic Processes*. This interdisciplinary collaboration focuses on the question whether and under which circumstances autonomous control may be more advantageous than classical control, especially regarding time, costs and robustness. The guiding principle of autonomous units is the possibility to integrate autonomous control into the model of the processes. This provides a framework for a semantically sound investigation and

comparison of different mechanisms of autonomous control. In more detail the aims are the following:

1. Algorithmic and particularly logistic processes shall be described in a very general and uniform way, based on a well-founded semantics.
2. The range of applications and included methods should comprise methods starting from classical process chain models like the one by Kuhn (see, e.g., (Kuhn 2002)) or Scheer (see, e.g., (Scheer 2002)) and the well-known Petri nets (see, e.g., (Reisig 1998)) leading to multi-agent systems (see, e.g., (Weiss 1999)) and swarm intelligence (see, e.g., (Kennedy and Eberhart 2001)).
3. The fact that autonomous units are based on rules provides the foundation for the dynamics of the processes. The application of these rules causes local changes in the common environment, yielding the steps of the processes, transformations, and computations. Archetypes for this behavior are grammatical systems of all kinds (see, e.g., (Rozenberg and Salomaa 1997)) and term rewriting systems (see, e.g., (Baader and Nipkow 1998)) as well as the domain of graph transformation (see, e.g., (Rozenberg 1997; Ehrig et al. 1999a; Ehrig et al. 1999b) and DNA computing (see, e.g., (Păun et al. 1998)). The rule-based approach is meant to ensure the possibility of executing the semantics as well as to lay the foundation for formal verification.
4. The autonomous control should become apparent on two levels. On the one hand a system comprises a community of autonomous units in an underlying environment. On this level all the units are considered equal in the sense that they may act independently of other units (provided that the state of the environment is suitable for the application of the desired rules). Since no further control exists, the units act autonomously. On the other hand transformation units as rule-based systems are typically nondeterministic, since at any time several rules may be applicable, or the same rule may even be applicable at different positions. In this case the autonomous control facilitates the selection of the different possibilities.

The following section introduces autonomous units. In Sects. 3 to 5 the semantics of a community of autonomous units is defined in three stages. First of all a simple sequential semantics is introduced. This semantics is merely suitable for systems that allow only one action at a time. This covers not only many algorithms and sequential processes, but also card and board games. The sequential semantics of autonomous units is also presented and investigated in (Hölscher et al. 2006b). On the second stage a

parallel semantics is defined. Here a number of actions take place in parallel at the same time. This allows for an adequate description of parallel derivations in L-systems (see e.g., (Rozenberg and Salomaa 1998), the firing of Petri nets, and parallel algorithms and processes. While the parallel actions in this semantics occur sequentially, the third stage defines a concurrent semantics with no chronological relations between the acting units. Here the autonomous units may act independently, unless a causal relationship demands a certain order of actions.

The concept of autonomous units is illustrated employing two examples. On the one hand place-transition systems are modeled so that every transition corresponds to one autonomous unit. On the other hand a transport network with packages and trucks is described as a system of autonomous units. Here every package as well as every truck is modeled as an autonomous unit. The paper ends with a short conclusion.

It should be pointed out that autonomous units generalize the concept of transformation units, which has been investigated in e.g. (Janssens et al. 2005; Kuske 2000; Kreowski and Kuske 1999; Kreowski et al. 1997). Here the derivation process is controlled by a main transformation unit and no changes of the environment can occur outside of this control. First steps towards distributed transformation units can be found in (Knirsch and Kuske 2002).

Autonomous Units

In this section, the concept of autonomous units is introduced as a modeling approach for data processing systems with autonomous components. Autonomous units form a community with a common environment, which they may transform.

For the sake of simplicity we represent the environments as graphs. But graphs are used in a quite generic sense, including all sorts of diagrams. They may be directed, undirected, labeled or attributed. Since graphs may comprise different subgraphs and different connected components it is also possible to use sets, multisets, and lists of graphs or even arbitrarily structured graphs as environments.

Every autonomous unit is equipped with a goal, rules and a control condition, which autonomously manages the application of the rules in order to accomplish the given goal. Rules transform the environment through their application, thus defining a binary relation of environments as their semantics. Since the control condition determines which rules may be applied to the current environment, its semantics is also defined as a binary relation of environments. Goals are formulated as class expressions, the

semantics of which is a class of environments in which the goal is accomplished.

All available environments, rules, control conditions and class expressions form a transformation approach. Its rules, control conditions and class expressions provide the syntactical ingredients of autonomous units. Additionally class expressions are used to define the initial environments and the overall goals of system models.

A *transformation approach* $\mathcal{A}=(\mathcal{G}, \mathcal{R}, \mathcal{X}, \mathcal{C})$ consists of a class of graphs $\mathcal{G}$, called *environments*, a class of *rules* $\mathcal{R}$, a class of *environment class expressions* $\mathcal{X}$ and a class of *control conditions* $\mathcal{C}$. Every rule $r \in \mathcal{R}$ specifies a binary semantic relation $SEM(r) \in \mathcal{G} \times \mathcal{G}$. Every pair $(G,H) \in SEM(r)$ is a rule application of r, which is also called a *direct derivation* and denoted as $G \Rightarrow_r H$. The semantics of a class expression $X \in \mathcal{X}$ is specified as a set $SEM(X) \subseteq \mathcal{G}$ of environments. A control condition defines a binary relation $SEM(C) \subseteq \mathcal{G} \times \mathcal{G}$ on environments as semantics.

A *community of autonomous units* $COM=(Goal,Init,Aut)$ consists of an environment class expression *Goal*, defining the *terminal environments* and thus the *overall goal*, an environment class expression *Init*, specifying the *initial environments*, and a set *Aut* of autonomous units.

An *autonomous unit* is a tuple $aut=(goal,rules,control)$ with $goal \in \mathcal{X}$ being the individual *goal*, $rules \subseteq \mathcal{R}$ being the *set of rules*, and $control \in \mathcal{C}$ being the *control*.

Example: Place/Transition Systems

Place/Transition (*P/T*) systems are a frequently used kind of Petri nets that can be modeled as a system of autonomous units. The *P/T* net with its marking is regarded as the environment. Transitions are modeled as rules. The firing of a transition defines a rule application that changes the marking in the usual way. Class expressions may be single markings, which define themselves as semantics. A further class expression *all* is also needed, meaning that all environments are permitted as goals. The control condition consists solely of the standard condition *free*, which defines all pairs of environments and imposes no restrictions on the application of rules.

If every transition t is considered as an autonomous unit $aut(t)=(all,\{t\},free)$ a *P/T* net N with the set of transitions T and initial marking m_0 is modeled as the community of autonomous units $COM(N,m_0)=(all,m_0,\{aut(t) \mid t \in T\})$.

Example: Transport Net

As a further illustration, a simplified example from the domain of transport logistics is sketched.

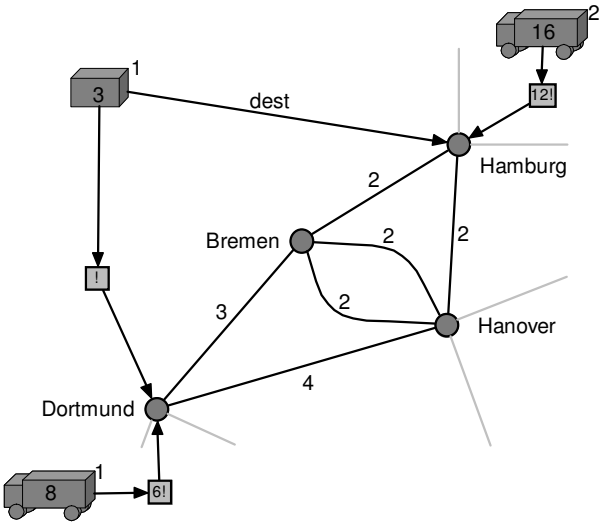

Fig. 1. A transport net represented by a graph

A transport net is a graph in which nodes represent locations, e.g. depots, where packages may be picked up and to which packages may be delivered. The edges represent the connections between the depots. Every edge is labeled with the time that is needed to travel along the connection that is represented by the edge. Fig. 1 shows a small excerpt of a transport net containing depots in the cities Dortmund, Bremen, Hamburg and Hanover. Trucks and packages are modeled as autonomous units, which use the transport net as underlying environment. Instances of these autonomous units are represented as special nodes with unique identifiers. The transport net contains two trucks (1, 2) and one package (1). The truck nodes are labeled with a number, which represents the amount of time the truck may be moving around. In the given example truck 1 is permitted to move around eight hours, while truck 2 may move around 16 hours (because it may be equipped with two drivers). Both truck nodes are connected to a rectangular tour node which is labeled with a number and an exclamation mark. The number defines the payload capacity of the truck, in our example specified in tons. Truck 1 has a payload of 6 tons,

and truck 2 may load up to 12 tons of cargo. The exclamation mark indicates the current tour node. A package node is labeled with a number which specifies its weight. It is also connected to a rectangular tour node, which in turn is connected to the depot that currently holds the package. Analogously to the truck tour node the exclamation mark indicates the current package tour node. An edge labeled "dest" connects the package node with its destination depot, i.e. the depot to which the package has to be delivered.

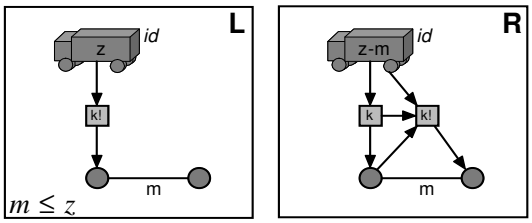

Fig. 2. Arranging a truck tour

The transformation unit *truck* contains a rule for planning a tour. This rule is depicted in Fig. 2. The application of this rule extends the current truck tour. This is done by adding a tour section leading from the current depot to an adjacent depot. Here the remaining travel time z of the truck must be at least as great as the travel time m between the depots, denoted by the application condition $m \leq z$. Such an application condition has to be evaluated to true, otherwise the rule may not be applied. The application of the rule defines the newly added tour section (represented by the added tour node) as current, and reduces the travel time of the truck by the time that is needed to drive to the adjoining depot.

A package unit has a tour planning rule that is similar to the rule of the truck units. It is depicted in Fig. 3.

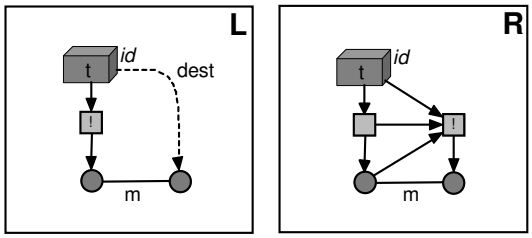

Fig. 3. Planning a package tour

The application of this rule extends the package tour by adding a new package tour node and connecting it to an adjacent depot. Analogously to the truck rule the newly added package tour node becomes the current one. This rule should only be applicable if the package is not planning its final tour section. This is modeled in the left-hand side of the rule by the dashed edge connecting the package node with a depot. This edge is labeled with "dest", indicating that the depot is the place to which the package should be delivered. The dashed edge is called a negative application condition (*NAC*) (Habel et al. 1996). If a situation as specified in the NAC is present in the transport net, the rule cannot be applied. Hence, the rule must not be applied if the adjacent depot is already the target depot of the package.

If this is the case the second tour planning rule of the package unit is needed. It is depicted in Fig. 4.

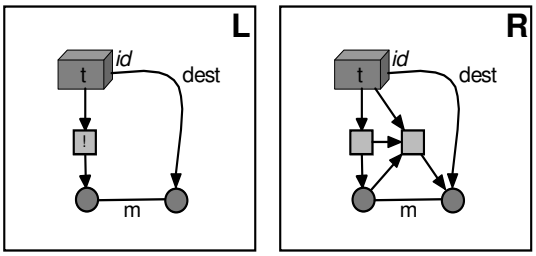

Fig. 4. The final part of the package tour

Here the adjacent depot must be the destination depot of the package, as indicated by the edge labeled with "dest". Basically the application of the rule yields the same changes as the first tour planning rule of the package unit. The only difference is that the newly added package tour node is not labeled with an exclamation mark. This is due to the fact, that no current tour node is needed anymore, because the package has finished its tour planning. Given these tour planning rules, truck units as well as package units may independently plan their tours.

After planning its tour a package should be picked up by a truck. Therefore, a package unit contains a rule that makes an offer to a passing truck. This rule is depicted in Fig. 5. The rule may be applied if a tour section of a truck coincides with a tour section of the package and the payload capacity k of the truck for this tour section is sufficient for the transport of the package (as indicated by the application condition $t \leq k$). The application of the rule inserts a new edge into the environment, connecting the package tour node to the truck tour node. It is labeled with the actual offer n and a

question mark, indicating that an offer for transportation has been made for the amount of n currency units.

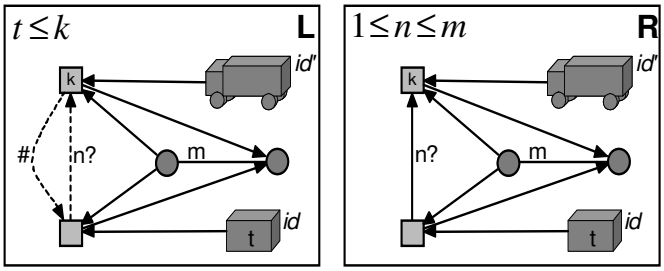

Fig. 5. A package offer

The dashed edges in the left-hand side **L** of the rule again represent negative application conditions. They guarantee that no previous offer was made by the package to the truck if one of the dashed edges with the given labels connect the tour nodes in the specified way. No offer can be made if either the package unit has already made an offer with some amount n, or if the truck unit has finally rejected the offer (indicated by the label "#"). The right-hand side of the rule also contains the post-condition $1 \leq n \leq m$. Such a post-condition has to hold after the rule is applied. In this case, the post-condition guarantees that the package will always offer an amount that is proportionally related to the distance.

The truck unit contains two rules which handle package offers. The first rule is depicted in Fig. 6.

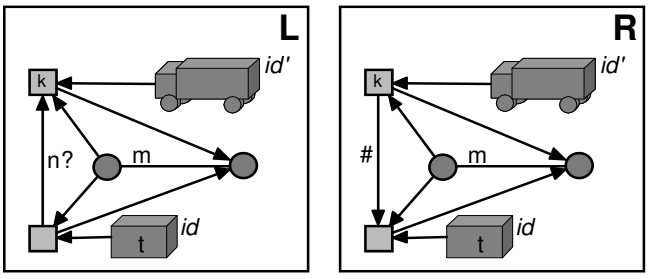

Fig. 6. Refusal of an offer

It specifies the rejection of a package offer by deleting the edge representing the offer and inserting a reversely directed edge labeled "#". In this

case a package unit cannot make another offer, because the NAC of the offer rule prohibits the existence of such an edge. The second rule is depicted in Fig. 7 and specifies the acceptance of an offer.

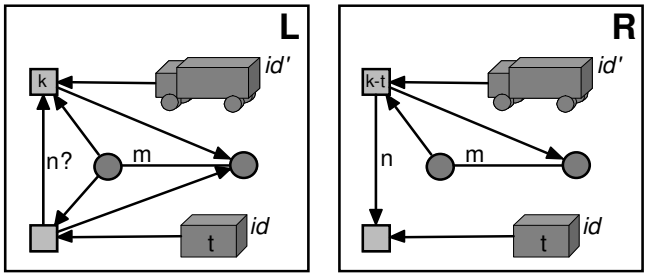

Fig. 7. Accepting a package

Similar to the first rule, the edge representing the offer is removed and a reversely directed edge is inserted. But in this case the edge is labeled with *n*, indicating that the truck transports the package in this section of its tour for a payment of *n*. Additionally the weight *t* of the package is subtracted from the payload capacity *k* of the truck for the corresponding tour section. The connections of the package tour node to the depots are removed, since the actual route of this tour section is described by the tour node of the truck. This removal also ensures that the package does not make any further offers for this tour section.

In the following sections the semantics of communities of autonomous units is defined in three variants. The simplest one is the sequential semantics, which is merely suitable for systems that allow only one action at a time. The parallel semantics allows for activities to take place in parallel, i.e. in a synchronized way. The third variant covers true concurrency. Only causally related activities (e.g. one action needs something that is created by another action) occur in chronological order. Other activities may happen at any time.

Sequential Semantics

Since the application of rules provides single computational steps, a first simple semantics for communities of autonomous units is obtained by sequential composition of these steps. This yields finite as well as infinite sequential processes.

Let $COM = (Goal, Init, Aut)$ be a community of autonomous units. A *finite sequential process*, also called *derivation* or *computation* is then defined by $(G_i)_{i\in[n]}$ with $[n]=\{0,\ldots,n\}$ for $n \in \mathbb{N}$, where the following holds for all $i = 1,\ldots,n$:

An autonomous unit $aut_i = (goal_i, rules_i, control_i)$ and a rule $r_i \in rules_i$ exist such that $G_{i-1} \Rightarrow_{r_i} G_i$ and $(G_{i-1}, G_i) \in SEM(control_i)$.

Analogously, an *infinite sequential process* is given by a sequence $(G_i)_{i\in\mathbb{N}}$ with the same properties as in the finite case, but for $i \in \mathbb{N}$. In this sense processes are arbitrary sequential compositions of rule applications by autonomous units, obeying the control condition of the currently active unit. The set of all sequential processes is denoted as $SEQ(Aut)$. Accordingly, $SEQ(Init,Aut)$ contains all processes which start with an initial environment, and $SEQ(Goal,Init,Aut) = SEQ(COM)$ contains all finite processes which additionally terminate in an environment that meets the goal.

In the latter case the semantics can also be defined by an input-output relation, which describes the computation without intermediate steps: we have $(G,H) \in REL_{SEQ}(COM)$ if $(G_i)_{i\in[n]} \in SEQ(COM)$ exists such that $G=G_0$ and $H = G_n$. Even for arbitrary processes the goal specification makes sense, since it can be determined whether *Goal* has been reached for processes $(G_i)_{i\in\mathbb{N}}$ in intermediate steps: $G_{i_0} \in SEM(Goal)$ for some $i_0 \in \mathbb{N}$?
Analogously, a sequential semantics for a single autonomous unit $aut = (goal,rules,control)$ can be defined taking into account that besides the rule application of the considered unit other units may also change the environment.

Let $CHANGE \subseteq \mathcal{G} \times \mathcal{G}$ be a binary relation on environments, which describes all changes that are not performed by *aut*. Let further $N = [n] = \{0,\ldots,n\}$ for an $n \in \mathbb{N}$ or $N = \mathbb{N}$. Then a sequential process of *aut* is a sequence $(G_i)_{i\in\mathbb{N}}$ such that for all $i \geq 1$ either $(G_{i-1}, G_i) \in CHANGE$ or for an $r \in rules$ $G_{i-1} \Rightarrow_r G_i$ and $(G_{i-1}, G_i) \in SEM(control)$. The set of sequential *aut* processes is denoted as $SEQ_{CHANGE}(aut)$.

The sequential processes $SEQ(Aut)$ of a set *Aut* of autonomous units and the sequential processes of one of its members are strongly connected:

$$\mathrm{SEQ(Aut)} = \mathrm{SEQ}_{\mathrm{SEQ(Aut-\{aut\})}}(\mathrm{aut}).$$

So every sequential process is an *aut* process for every autonomous unit in *Aut* and vice versa, provided that the changes in the environment are precisely the sequential processes of the other autonomous units.

Example Place/Transition Systems

Let $COM(N,m_0)$ be the system of autonomous units that corresponds to a *P/T* system. Then the application of a rule yields the same effect as the firing of a transition. In this way sequential processes correspond to the firing sequences of the *P/T* system.

Example Transport Net

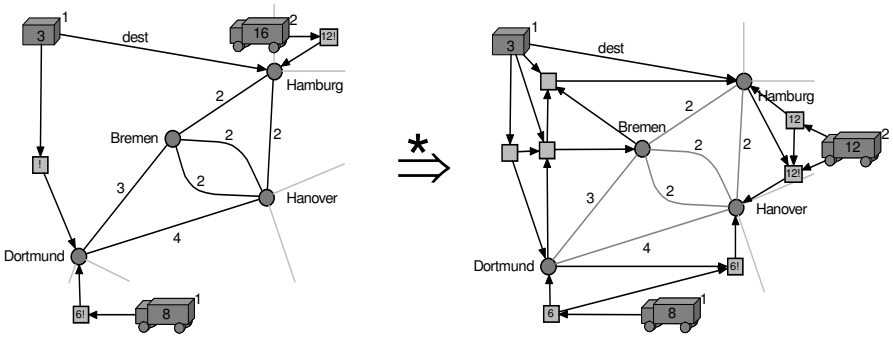

Fig. 8. Derivation

The planning of package and truck tours can be regarded as sequential processes in a transport net. Fig. 8 depicts a process for the tour planning of package 1, which intends to be transported from Dortmund to Hamburg via Bremen, while truck 1 and truck 2 each planned a tour to Hanover, with truck 1 originating in Dortmund and truck 2 starting in Hamburg.

Parallel Semantics

In many cases it is rather unrealistic to consider a system of autonomous units that transform the shared environment in a sequential way. The actual processes in most data processing systems are more suitably modeled by allowing more than one activity on the environment at the same time. This includes in particular the fact that different events which do not influence each other, can happen in parallel.

In order to obtain a suitable formal definition of parallel processes it is necessary to extend the assumptions on the given graph transformation approach. So far we have considered situations where only one rule is applied at a time. For the parallel semantics definition let us now consider situations where a multiset of rules may be applied to the environment.

This means that a number of different rules may be applied or even a single rule may be applied multiple times. For this purpose let $\mathcal{A}=(\mathcal{G}, \mathcal{R}, \mathcal{X}, C)$ be a *parallel transformation approach*, meaning that a binary semantic relation $SEM(R) \subseteq \mathcal{G} \times \mathcal{G}$ exists for every multiset R of $\mathcal{R}$. Instead of $(G,H) \in SEM(R)$ this may also be denoted as $G \Rightarrow_R H$ and may also be called direct parallel derivation.

Parallel processes are then defined analogously to the sequential case. First the occurrence of a single rule application has to be replaced by the application of a multiset of rules. Secondly the definition for obeying the control condition has to be changed. The corresponding sets of parallel processes $PAR(Aut)$, $PAR(Init,Aut)$, and $PAR(Goal,Init,Aut) = PAR(COM)$ as well as an input-output relation $REL_{PAR}(COM)$ are then obtained analogously to the sequential case.

Example Place/Transition Systems

A parallel transformation approach is obtained by defining parallel firing of a multiset of transitions in a *P/T* system in the usual way. For the system $COM(N,m_0)$ the parallel processes correspond exactly to the firing sequences of multisets of transitions.

Example Transport Net

The tour planning of package and truck units can also be regarded as parallel processes in a transport net. The process depicted in Fig. 8 can be modeled as a parallel procedure of one package tour planning step and both truck tour planning steps, followed (or preceeded) by the remaining package tour planning step.

In general, sequential and parallel processes may produce very different results. Consider for instance cellular automata, where a transition step of all linked finite automata depends on the state of their neighbors. Here a parallel computational step of some automata would change the context of the other automata such that later steps yield different configurations. In other approaches, like e.g. Petri nets, term replacement, or most approaches to graph transformation, parallel changes do not affect the final output, but yield a reduced number of transformation steps. This is due to the fact that the parallel actions may also occur sequentially in an arbitrary order without affecting the final result. This phenomenon is called *true concurrency*. In order to obtain true concurrency in the context of parallel transformation approaches the following has to hold:

Let $R = R' + R''$ be the sum of two multisets of rules and $G \Rightarrow_R X$ be a parallel derivation step. Then parallel derivation steps $G \Rightarrow_{R'} H$ and $H \Rightarrow_{R''} X$ exist for a suitable environment *H*.

Remember that every multiset is the commutative sum of its single elements. For this reason true concurrency implies that every parallel step could also be executed as an arbitrarily ordered sequence of the corresponding single rule applications, yielding the same result. Parallel processes and their sequentialization are called *equivalent* in the context of concurrency. Consider an equivalence class of a parallel process, i.e. all processes that are equivalent to each other. Then the chronological order of two rule applications can only be determined if the one causally depends on the other. Otherwise they can be applied in parallel or in an arbitrarily ordered sequential way.

Since every sequential transformation step is a special case of a parallel transformation step, the sequential semantics of autonomous units is contained in the parallel semantics, i.e. $SEQ(A) \subseteq PAR(A)$ is true for the processes of a set of autonomous units *A*. Furthermore an equivalent process $\bar{s} \in SEQ(A)$ can be found for every process $s \in PAR(A)$. For a system of autonomous units *S* this implies in particular

$$REL_{SEQ}(S)=REL_{PAR}(S).$$

Concurrent Semantics

Like the sequential process semantics, the parallel process semantics may not be suitable for every application situation. This is due to the fact that components which act autonomously and independently, do not necessarily start and finish their activities simultaneously, as is the case with parallel steps. If such components act far away from each other, or work on completely different tasks without influencing each other it may even not be possible to determine simultaneity. Anyway, demanding or enforcing simultaneity would not make any sense in this case. A chronological order of concurrent and distributed processes is only given in the case that one activity needs something that another activity provides. Such causal relationships can be expressed by directed edges between these activities. In the case of concurrent processes this results in an acyclic graph of activities. Such a graph yields a concept for concurrent processes in communities of autonomous units. This is basically the same idea as in the notion of processes of Petri nets.

Let $COM = (Goal, Init, Aut)$ be a system of autonomous units over a parallel transformation approach $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \mathcal{X}, C)$. Then a *concurrent process* consists of an initial environment G_0 and an acyclic, directed graph $run=(V,E,lab)$, with a set of nodes V and a set of edges $E \subseteq V \times V$. The nodes are marked with $lab: V \rightarrow \mathcal{R}$, which maps every node to a rule. The following must also hold for G_0 and *run*:

1. Every node in *run* must be reachable via a path originating in an initial node, i.e. a node without incoming edges.
2. Every complete beginning part of *run*, i.e. every subgraph which contains all initial nodes and with every node also all paths from the initial nodes to that node, is either finite or contains an infinite path.
3. For every complete beginning part a parallel process $(G_i)_{i \in N}$ together with a bijection between the nodes of the subgraph and the applied rules can be found for $N = [n]$, $n \in \mathbb{N}$ or $N = \mathbb{N}$. These rules conform to the markings of the nodes. This bijective relation keeps the causal dependency. This means that a rule which marks the source of an edge in the subgraph is always applied in an earlier step than the rule which marks the target of this edge.

The first condition enforces that *run* does not contain infinite paths without start. Otherwise there would be a path with no corresponding process. The second condition implies that only finitely many nodes are causally independent of each other. The third condition guarantees that concurrent processes are actually executable.

Example Place/Transition Systems

With the notion of occurrence nets at least the special case of *Condition/Event*(*C/E*) nets has a similar process concept. If every path of length 2 that runs along a condition is replaced with a directed edge in such an occurrence net, then a concurrent process in the aforementioned sense is obtained.

Example Transport Net

In the transport net example the tours of trucks as well as packages can be planned concurrently. The negotiation for transport of different truck-package pairs may also occur in a concurrent way.

An elaborated description of the relation between parallel and concurrent processes goes beyond the scope of this paper and thus has to be de-

ferred to further work. But it is noteworthy to mention that in the case of true concurrency a strong relation between concurrent processes and canonical derivations exists. This has been investigated in (Kreowski 1978) in the context of graph transformation employing the double-pushout approach. Such canonical derivations represent equivalence classes of parallel derivations in a unique way by enforcing maximum parallelity and an application as soon as possible.

Conclusion

In this chapter we have introduced the new concept of autonomous units. This rule-based concept is meant to model data processing systems comprising different distributed components and processes. These components may act autonomously but they may also communicate and interact with each other. The operational semantics of such systems has been defined in three stages. Sequential and parallel processes establish a chronological order of the activities in such a system. In the context of concurrent processes only the chronological order of causally related activities is fixed. The approach employs graphs and graph transformation rules allowing visual models, as illustrated by the example of transport nets. At the same time the concept is flexible by allowing to embed different modelling approaches, which provides the opportunity of semantical comparisons between different modeling methods. The example of Petri nets gives a hint in this direction, which has to be substantiated by further research in future work. Anyhow a number of aspects have not been addressed so far and some questions have been left unanswered in this introductory work. This includes, among others, the following:

1. The sequential and parallel processes are composed of application of either single rules or multisets of rules. This is closely related to the semantics of labeled transition systems, which are frequently used for the semantic foundation of communication and distributed systems. This relation demands further investigation.
2. As indicated at the end of Sect. 5, a strong relation exists between concurrent processes and canonical derivations, the latter being special kinds of parallel processes. The detailed investigation of this relationship would be interesting.
3. A remarkable aspect of the classical transformation units is the structuring principle. This is achieved by the import feature of transformation units, which allows them to import other transformation units and utilize them to solve subtasks. So far only autonomous units with sequential semantics have been defined with an additional structuring

principle (cf. (Hölscher et al. 2006b)). But it would generally make sense for autonomous units to modularize the solution of a task or to let subtasks be handled by other autonomous units. For this reason, future work should also concentrate on structured autonomous units in the parallel and concurrent cases.

4. The main task for further investigation of autonomous units will be to investigate the means of control. On the one hand specific control mechanisms allowing for autonomy have to be investigated. This will comprise in particular concepts for the evaluation of the environment and for a more goal oriented control. On the other hand, the control, which is currently only defined for single steps, has to be enhanced to also cover extended processes, as this is already the case with classical transformation units and sequential autonomous units.
5. The significance and suitability of autonomous units as a modeling approach will be proved by a number of case studies. These will comprise studies reaching from games over agent systems and artificial ant colonies to the conventional approaches of process modeling. A first example can be found in (Hölscher et al. 2006b), where the board game ludo is modelled with autonomous units.
6. A further theoretical investigation of autonomous units together with existing theoretical results would be useful for the practical application of autonomous units. This includes for example decidability results of control conditions (Hölscher et al. 2006a) or class expressions as well as (automated) correctness proofs.

References

Baader F, Nipkow T (1998) Term Rewriting and All That. Cambridge University Press, Cambridge

Ehrig H, Engels G, Kreowski H-J, Rozenberg G (eds) (1999) Handbook of Graph Grammars and Computing by Graph Transformation, vol 2: Applications, Languages and Tools. World Scientific, Singapore

Ehrig H, Kreowski H-J, Montanari U, Rozenberg G (eds) (1999) Handbook of Graph Grammars and Computing by Graph Transformation, vol 3: Concurrency, Parallelism, and Distribution. World Scientific, Singapore

Janssens D, Kreowski H-J, Rozenberg G (2005) Main Concepts of Networks of Transformation Units with Interlinking Semantics. In: Kreowski H-J, Montanari U, Orejas F, Rozenberg G, Taentzer G (eds) Formal Methods in Software and System Modeling, Lecture Notes in Computer Science vol 3393. Springer, Berlin Heidelberg New York, pp 325-342

Habel A, Heckel R, Taentzer G (1996) Graph Grammars with Negative Application Conditions. Fundamenta Informaticae, 26:287-313

Hölscher K, Klempien-Hinrichs R, Knirsch P (2006a) Undecidable Control Conditions in Graph Transformation Units. In: Moreira Martins A, Ribeiro L (eds) Brazilian Symposium on Formal Methods (SBMF 2006), pp 121-135

Hölscher K, Kreowski H-J, Kuske S (2006b) Autonomous Units and their Semantics – the Sequential Case. In: Corradini, A, Ehrig H, Montanari U, Ribeiro L, Rozenberg G (eds) Proc. 3rd International Conference on Graph Transformations (ICGT 2006), Lecture Notes in Computer Science vol 4178, Springer, Berlin Heidelberg New York, pp 245-259

Kennedy J, Eberhart RC (2001) Swarm Intelligence. Morgan Kaufmann Publishers, San Francisco

Kreowski H-J, Kuske S (1999) Graph Transformation Units with Interleaving Semantics. Formal Aspects of Computing 11(6):690-723

Knirsch P, Kuske S (2002) Distributed Graph Transformation Units. In: Corradini A, Ehrig H, Kreowski H-J, Rozenberg G (eds) Proc. First International Conference on Graph Transformation (ICGT), Lecture Notes in Computer Science vol 2505, Springer, Berlin Heidelberg New York, pp 207-222

Kreowski H-J, Kuske S, Schürr A (1997) Nested Graph Transformation Units. International Journal on Software Engineering and Knowledge Engineering, 7(4):479-502

Kreowski H-J (1978) Manipulationen von Graphmanipulationen. Ph.D. thesis, Berlin

Kuhn A (2002) Prozessketten - ein Modell für die Logistik. In: Wiendahl H-P (ed) Erfolgsfaktor Logistikqualität. Springer, Berlin Heidelberg New York, pp 58-72

Kuske S (2000) Transformation Units-A Structuring Principle for Graph Transformation Systems. Ph.D. thesis, Bremen

Păun G, Rozenberg G, Salomaa A (1998) DNA Computing-New Computing Paradigms. Springer, Berlin Heidelberg New York

Reisig W (1998) Elements of Distributed Algorithms-Modeling and Analysis with Petri Nets. Springer, Berlin Heidelberg New York

Rozenberg G (ed) (1997) Handbook of Graph Grammars and Computing by Graph Transformation, vol 1: Foundations. World Scientific, Singapore

Rozenberg G, Salomaa A (eds) (1997) Handbook of Formal Languages, vol 1-3. Springer, Berlin Heidelberg New York

Rozenberg G, Salomaa A (eds) (1998) Lindenmayer Systems. Springer, Berlin Heidelberg New York

Scheer A-W (2002) Vom Geschäftsprozeß zum Anwendungssystem. Springer, Berlin Heidelberg New York

Weiss G (ed) (1999) Multiagent Systems-A Modern Approach to Distributed Artificial Intelligence. The MIT Press, Cambridge, Massachusetts