Communities of Autonomous Units for Pickup and Delivery Vehicle Routing*

Hans-Jörg Kreowski and Sabine Kuske

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

Abstract. Communities of autonomous units are being developed for formal specification and semantic analysis of systems of interacting and mobile components. The autonomous units of a community are rule-based, self-controlled, goal-driven, and operate and move in a common environment. We employ communities of autonomous units to model the dynamic pickup and delivery problem with the general idea to demonstrate their suitability for a range of logistic tasks.

1 Introduction

Many recent approaches in computer science like, communication networks, multi-agent systems, swarm intelligence, ubiquitous, wearable, and mobile computing involve widely spread autonomous components that interact and communicate with each other, move around or connect themselves to other components to form networks. To cover such new programming and modeling paradigms in a formally well-founded and visually well-describable way we proposed autonomous units, as a rule-based, self-controlled, and goal-driven concept (see, e.g., [1]).

A system of autonomous units forms a community provided with a common environment where the units interact and may have an overall goal. The autonomous units of a community apply transformation rules to the common environment in a self-controlled and goal-driven manner. A transformation rule application may modify the environment, send messages to other autonomous units, react to received messages or to environment modifications performed by other units, connect and disconnect the unit to and from other units, or move the unit around the environment.

For this purpose an autonomous unit is composed of a set of transformation rules, a control component to regulate its rule application process and a goal that the unit tries to achieve. Moreover, a unit has a private state in which the unit can store private data and which can only be transformed by the unit itself. A transformation rule r may simultaneously transform the common environment

* Research partially supported by the Collaborative Research Centre 637 (Autonomous Cooperating Logistic Processes: A Paradigm Shift and Its Limitations) funded by the German Research Foundation (DFG).

A. Schürr, M. Nagl, and A. Zündorf (Eds.): AGTIVE 2007, LNCS 5088, pp. 281–296, 2008.
© Springer-Verlag Berlin Heidelberg 2008

and the private state of the unit. This means that r is a product rule consisting of two transformation rules: one for the common environment and the other for the private state (cf. [2]).

In order to keep the rule set of an autonomous unit readable and small, it can be structured hierarchically into transformation units each of which consists of a set of transformation rules and a control condition. These reusable transformation units perform actions that require the controlled application of several rules (cf. for example [3] and [4]).

In general, autonomous units can work in parallel. The operational semantics of a community consists of (perhaps never-ending) sequences of states such that every state is composed of the current common environment plus all current private states. A transformation from one state to the next happens if some or all units of the community apply one or more rules in parallel. Hence, state transformation consists of the parallel application of a set product rules.

Since environments can often be modeled and visualized as graphs, since graphs can be modified in a straightforward way by graph transformation rules, and since graph transformation has a precisely defined semantics [5] it seems to be natural to specify the actions of autonomous units with graph transformation rules. However, the concept of autonomous units is not restricted to the graph transformational approach.

The aim of this paper is to illustrate and demonstrate the potential of autonomous units to model logistic applications by presenting a case study that models the basic operations of the dynamic pickup and delivery problem (see e.g. [6,7,8]) by a community of autonomous units. To keep the paper technically simple, we introduce autonomous units in a rather informal way. Formal descriptions can be found in [4,9]. Nevertheless, it is worth noting that autonomous units as presented in the following are more sophisticated than those of previous papers because private states have not been considered before.

The development of autonomous units has its origin within the Collaborative Research Centre *Autonomous Cooperating Logistic Processes: A Paradigm Shift and Its Limitations* in which we investigate in an interdisciplinary way how self-controlled units can be successfully employed for logistic applications with the aim to get better results concerning time, costs and robustness (see also [10]). The central idea of autonomous units is to introduce self-control explicitly into the modeling of (logistic) processes in order to create a semantically well-founded framework in which different self-control-based mechanisms become comparable (cf. [1]).

The paper is organized as follows. Section 2 briefly recalls basic concepts, like graphs, graph transformation rules, graph class expressions and control conditions. Section 3 is dedicated to the private states and common environments of communities. In Sections 4 it is shown how the behavior of autonomous units can be modeled with product rules. Section 5 illustrates how the actions and interactions occuring in the dynamic pickup and delivery problem can be modeled with a community. Section 6 presents the semantics of autonomous units based

on which some correctness results concerning the case study are formulated. Section 7 discusses related work. The paper ends with the conclusion.

2 Basic Concepts

The basic components of autonomous units, namely graphs, graph transformation rules, graph class expressions, and control conditions are taken from an underlying graph transformation approach. In this section we present an instance of a graph transformation approach that will be used throughout this paper. Further examples and formal definitions can be found in e.g. [5,3].

Graphs. A graph consists of a set of labeled or unlabeled nodes and a set of labeled or unlabeled edges such that every edge connects two nodes. An edge can be directed or undirected. Nodes may be depicted in different shapes illustrating in this way the entities they represent. Fig. 5 shows an example of a graph where the houses, trucks, and rectangles are the nodes and the arrows are the directed edges. The houses are labeled with letters, the rectangles and the edges between houses with natural numbers whereas the trucks and the edges from rectangles to houses are unlabeled.

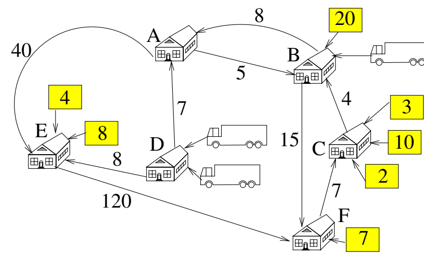

Fig. 1. Example of a graph

Graph transformation rules. A graph transformation rule consists of three graphs: a left-hand side, a right-hand side and a common part. An example of a rule is depicted in Fig. 2. The left-hand side is a house and the right-hand side consists of the same house, a rectangle labeled with m and an edge pointing from the rectangle to the house. The common part is the house, because it has the same number in the left- and the right-hand side of the rule. Often, the numbers of common nodes are not depicted. In this case the common items of a rule consist of the nodes and edges that occur in the left- as well as in right-hand side in the same relative positions. Please note that if the label m of the right-hand side is a variable of type $\mathbb{N}$ the rule represents a set of rules: one for each value in $\mathbb{N}$. These so-called parameterized rules will be often used in the following.

The application of a rule to a graph comprises the following steps: (1) Choose an image of the left-hand side in the graph. (2) Delete everything of this image that does not belong to the common part. (3) Glue the right-hand side into

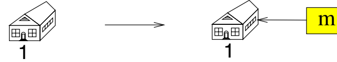

Fig. 2. Example of a rule

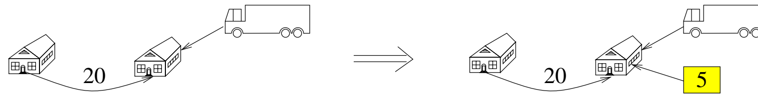

Fig. 3. A rule application

the graph such that the items of the common parts are identified with their images. Fig. 3 shows the application of the above rule after having substituted the variable m with the number 5. The rule is applied to a graph consisting of two houses that are connected by a directed edge labelled with 20, and a truck that is connected to the right house. The application of the rule adds a rectangle labeled with 5 to the right house. Clearly, the rule could also be applied such that the rectangle would be connected to the left house.

Graph class expressions. A graph class expression specifies a set of graphs. We use as graph class expressions graphs with variables of type $\mathbb{N}$ as node labels and a special form of graph grammars. More precisely, a graph with variables of type $\mathbb{N}$ as node labels specifies the set of all graphs that can be obtained by substituting each variable with a value of $\mathbb{N}$. For example, the right-hand side of the rule in Fig. 2 is a graph class expression of this kind. A graph grammar is a pair $GG = (S, P)$ where S is a graph called the *start graph* and P is a set of graph transformation rules. The set of graphs specified by GG consists of all graphs that can be generated by applying rules of P in a successive way starting from the start graph S.

Control conditions. A control condition is any expression that specifies a set of sequences of graphs. In this paper we use priorities and the special condition *free*. Given a set P of rules, a priority is a partial order $\leq$ on P and it specifies all sequences s of graphs such that for $i = 1, \ldots, n$ if $s = (G_1, \ldots, G_n)$ and for $i \in \mathbb{N}$ if s is infinite, G_i is obtained from G_{i-1} by applying a rule $p \in P$ and there is no rule $p' \in P$ with $p' > p$ that is applicable to G_{i-1}. In other words, this control condition allows to apply a rule to the current graph whenever there is no rule of a higher priority applicable. The condition *free* is the special case where all rules have the same priority.

3 Common Environments and Private States

Autonomous units act and interact within a common environment. In many cases an environment can be modeled as a graph in which certain nodes represent instances of autonomous units. In our case study of the pickup and delivery problem

the environment contains nodes representing trucks, customers and packages and the behavior of each of these nodes is specified by an instance of an autonomous unit.

Every community starts to work in an initial environment specified by a graph class expression. If one takes houses as customers and rectangles as packages, the graph in Fig. 1 is an initial environment of our case study. The set of all initial environments of the case study can be visually specified in a rule-based form by the graph grammar consisting of the rule set in Fig. 4 plus the empty graph (containing neither nodes nor edges) as start graph. The first rule generates customers. It contains a negative application condition [11] in its left-hand side which means that an A-labeled customer can only be generated if there doesn't exist one with the same label. In this way we make sure that all generated customers have different labels. The second rule connects different customers by edges labeled with a distance d of type $\mathbb{N}$. The application condition $A \neq B$ below the arrow requires that d-labeled edges be inserted between different customers, i.e. it avoids the insertion of loops at customers. The label d represents the time it lasts to move from the source customer to the target customer. If one wants to generate environments without parallel edges between customers, a convenient negative application condition could also be added to the second rule. The third rule inserts trucks and the fourth packages so that every truck is in the location of some customer and every package is offered by some customer. The label m is some natural number representing the weight of a package.

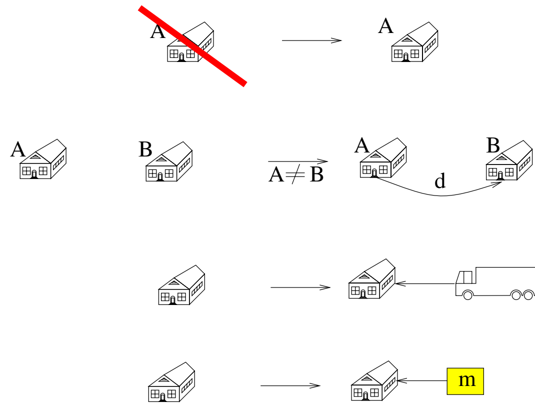

Fig. 4. Specification of the initial common environments

Obviously, trucks, customers and packages behave differently. Trucks, for example, can move, transport packages, plan their tours, etc. Packages select trucks for their transport, enter trucks and get out of them. Customers may offer or demand packages. As mentioned before, the behavior of these components is modeled with autonomous transformation units. Hence, after generating an initial environment, every truck node, every package node and every customer node is associated with (an instance of) the autonomous unit that specifies its

behavior. Technically, this can be achieved by adding a loop to every node v that is associated with a unit $type(v)$ and labeling this loop with $type(v)$. In the following every environment node v that is associated with an autonomous unit $type(v)$ is called the *local node* of $type(v)$.

Additionally to the common environment which can be transformed by all autonomous units of the community, every unit may have its own private state that can only be modified and seen by the unit itself. In this first approach, this private state contains the local node of the unit plus some additional information. For example, the private state of the autonomous unit *truck* stores its capacity, the weight of its current load and the weight of all packages which it has accepted to pickup later. Initially the latter two values are set to zero. The specification of the initial private states of the unit *truck* is depicted in Fig. 5 where the *max*-edge points to the capacity of the truck, the *w*-edge to the weight of the current load and the *r*- edge to the weight reserved for accepted packages. The reserved weight means the following. When a package asks a truck for being picked up the truck can accept this. In this case it reserves some weight (or place) in it for the package until the package enters the truck or until the truck starts to move.

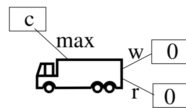

Fig. 5. Initial private state of *truck*

The common environment together with the private states form a set of graphs where each local node occurs in two copies: one in the common environment and one in the private state of the corresponding autonomous unit.

4 Modeling the Behavior of Autonomous Units

The autonomous units of a community may interact by transforming the environment, i.e. a change of the environment may be noticed by other units (re-)acting in the same community. Every autonomous unit *aut* that is associated with a node v in the environment specifies the behavior of v by means of some graph transformation rules, used transformation units, a control condition, a goal, and a private initial state containing the local node v plus some further private data. The rules of *aut* are split into common and private ones for transforming the common environment and the private state, respectively. Every rule r of *aut* that contains the node v in its left- and right-hand side should be applied in such a way that v is matched to the local node in the environment if r is a common rule and to the local node in the private state if r is private. In the following, every occurrence of v in a rule of *aut* is drawn with thick lines. In the rest of this section we show how autonomous units may communicate and change the common environment and private states with the use of graph transformation rules.

4.1 Interaction of Autonomous Units

A special form of interaction frequently used is *message sending*. This is modeled by the insertion of an edge labeled with the message content and going from the sender to the receiver. For example, if a package wants to enter into a truck it sends the message *enter?* to the truck. This can be modeled with the rule in Fig. 6 which belongs to the *package*-unit. It inserts an edge labeled with *enter?* from the local package to some truck that is at the same location and that will pass through the destination of the package. (Further details of the rule will be explained below.)

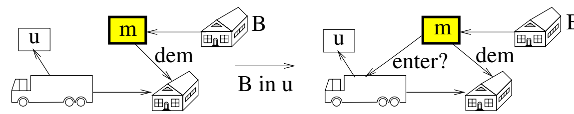

Fig. 6. Message sending

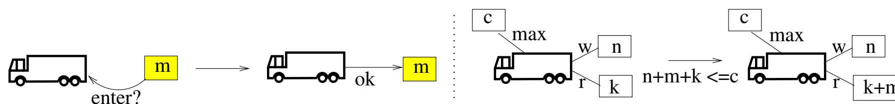

Fig. 7. Accepting a package

After receiving the *enter?*-message the truck can accept or reject to pickup the package. In case of acceptance the truck sends an *ok*-message to the package. This reaction is modeled with the left rule in Fig. 7. As one will see in Section 5, the truck changes its private state, simultaneously.

4.2 Modeling Behavior by Product Rules

Every autonomous unit can modify the environment by applying a graph transformation rule to it. Simultaneously, it can transform its private state to modify private data. This is achieved with the concept of product rules [2,12]. For our purpose we use a special form of product rules consisting of a pair $(com, priv)$ of rules which are applied simultaneously so that *com* modifies the common environment and *priv* the private state. In more detail, the application of $(com, priv)$ of a unit *aut* to a pair $(env, prist)$ consisting of a common environment *env* and a private state *prist* yields a pair $(env', prist')$ if env' can be obtained from *env* by applying *com*, and if the application of *priv* to *prist* yields $prist'$. As explained above, the rules must be applied in such a way that the local nodes be matched to the nodes associated with *aut*. This can be achieved with particular loops at the local nodes in rules, private states, and the common environment.

For example, if a truck accepts a package p, it reserves some of its capacity for this package. Hence, it applies the rule in Fig. 7 to the common environment and simultaneously, it adds the weight of the package to its reserved weight

by applying the right rule in Fig. 7. This rule (and hence the whole product rule) can only be applied if the transport of the package can be realized without exceeding the maximal capacity i.e. if $n+m+k \leq c$ where n is the current load of the truck, k is the reserved load, m is the weight of the package that is going to be accepted, and c is the maximal capacity of the truck. This application condition is denoted below the arrow of the private rule. Please note that c, n and k are variables that should be substituted with values when applying the rule.

It is worth noting that every product rule $(com, priv)$ of this special kind can be regarded as a triple graph transformation rule $(com, cp, priv)$ [13] where the left- as well as the right-hand side of the correspondence rule *cp* consists of the local node. One main difference between product rules and triple rules is that the former are approach independent while that latter are defined over a specific graph transformation approach. Moreover, product rules may have an arbitrary number of components rather than three ones as triple rules.

5 Pickup and Delivery with Autonomous Units

In this section, we describe how the basic operations of the dynamic pickup and delivery problem can be modeled with a community of autonomous units. The pickup and delivery problem consists of a set of customers, a set of vehicles (here trucks) and a set of packages. Basically, trucks move around in an environment to pickup and deliver packages thereby satisfying transport requests. Packages select trucks which they ask for being picked up and in case of acceptance they may enter into a truck and get out at their destination. Customers may offer or demand packages. In order to model the pickup and delivery problem conveniently, certain contraints must be satisfied such as time contraints or simply the requirement that the capacity of trucks should never be exceeded. The goal of every component (i.e. of every truck, customer, and package) is some objective function, like minimization of route length, costs, time, etc (cf. [6]).

The aim of this first approach towards modeling the pickup and delivery problem with autonomous units is to show how the basic operations of the dynamic pickup and delivery problem can be modeled based on graph transformation, so that trucks, packages and customers behave as autonomous entities in a common transport network where central control is dropped. A case study where the pickup and delivery problem is modeled with a single hierarchically structured transformation unit is presented in [14].

We assume in this stage of the case study that the goal of every autonomous unit is some objective function but we do not yet consider how it can be formulated in a graph transformational way and how control conditions can become goal driven. This will be studied in future work.

The basic behavior of the autonomous unit truck is specified in Fig. 8 where the parts *com* and *priv* of every product rule $(com, priv)$ are drawn side by side and with a dashed vertical line between them. As mentioned before the bold

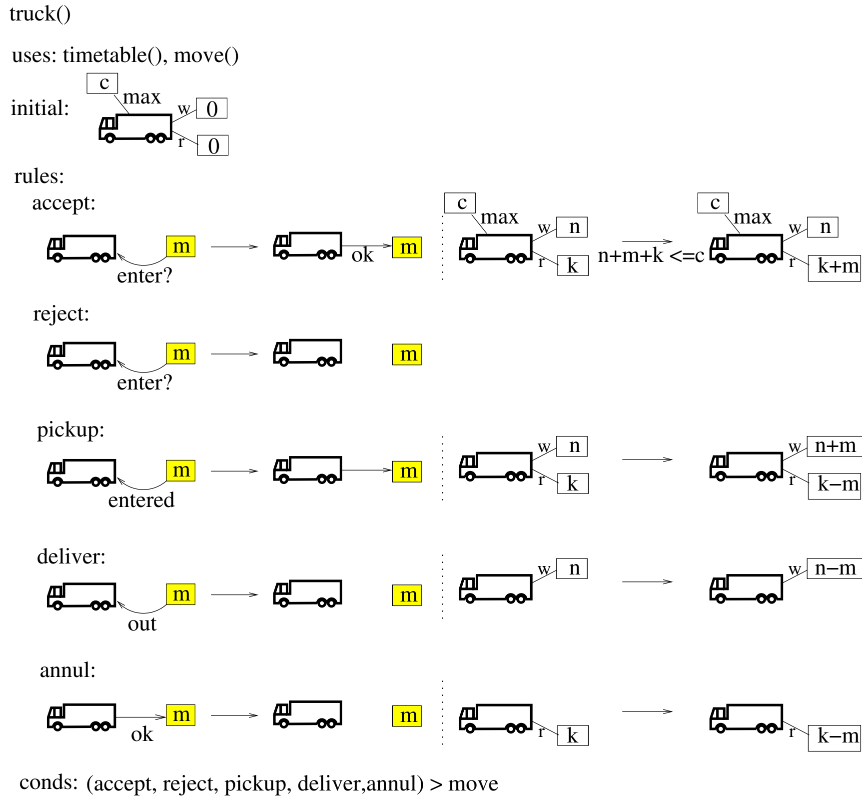

Fig. 8. The unit *truck*

nodes in the rules represent the local nodes of the unit. When applying a rule, these nodes must always be matched to the node associated with the unit which contains the rule.

The rules of the unit *truck* model interaction between trucks and packages from the point of view of the truck. As already explained in the previous section, the product rule *accept* can be applied if the truck has got a message *enter?* from some package. The application of the rule *accept* sends a message *ok* to the package and adds the weight of the package to the reserved load of the truck represented in the private state. Alternatively, the truck may reject the package by applying the second rule that deletes the *enter?*-edge. This rule does not modify the private state of the truck, i.e. the private part of the product rule is the empty rule and hence not depicted. The third product rule *pickup* can be applied when the truck receives an *entered*-message from a package. The edge from the truck to the package in the right-hand side models the fact that the package is in the truck. In the private rule of *pickup* the current weight of the truck is updated. The forth rule *deliver* can be applied if the truck has got an *out*-message from some package. When applying this rule, the truck deletes the *out*-egde and updates its current load. With the rule *annul* the truck can cancel

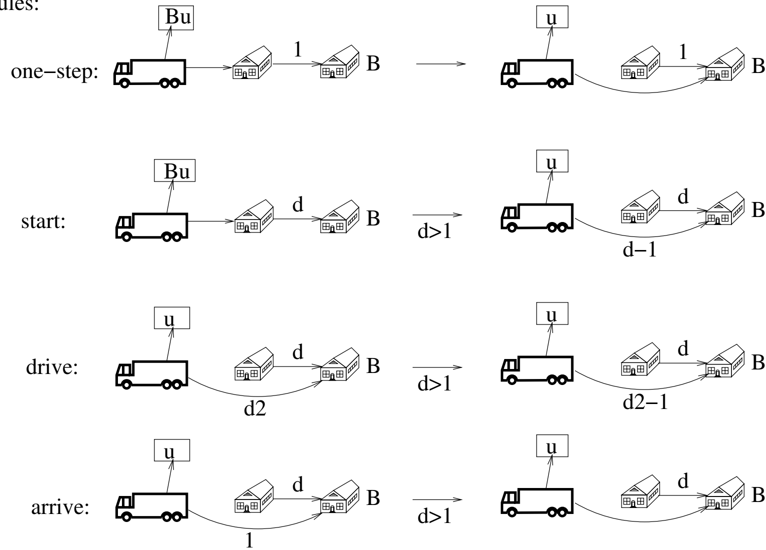

Fig. 9. The transformation unit *move*

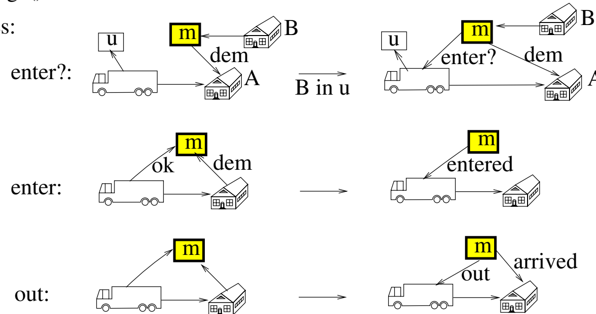

Fig. 10. The unit *package*

reservations. The imported unit *timetable* is not presented in detail. It links a node to the truck that is lableled with a string of customer names and which represents the tour the truck is going to move along. More precisely, a *tour* is a word $x_0 \cdots x_n$ of customer names such that for $i = 1, \ldots, n$ the customer x_{i-1} is connected to x_i through an edge.

The autonomous unit *truck* uses the transformation unit *move* depicted in Fig. 9. It models the movement of a truck from one customer to the next in the tour of the truck. The movement lasts exactly d steps (i.e. rule applications) if the edge has distance d.

The unit *package* is shown in Fig. 10. It contains three rules that modify the common environments. In the first rule the package wants to enter into a

truck which is at the same location as the package's owner A, provided that the package is demanded (denoted by the label *dem* at the edge from the package to A), and that the customer B who demands the package occurs in the route u of the truck. If the package gets an *ok*-message from a truck the former can decide with the second rule to enter the truck provided that the latter has not yet moved away. The application of the rule deletes the *dem*-edge from the package's owner. Hence, this rule can only be applied if the package is not on another truck. With the last rule a package can send an *out*-message to the truck provided that the package is in the truck and arrived at the customer who demanded it.

Please note that in this simplified case study the unit *package* has no private state. But in a further step we plan to include also a private state for packages that stores relevant information to choose a *good* truck (a cheap and fast one that transports the package safely within certain prescribed time windows) and not an arbitrary one.

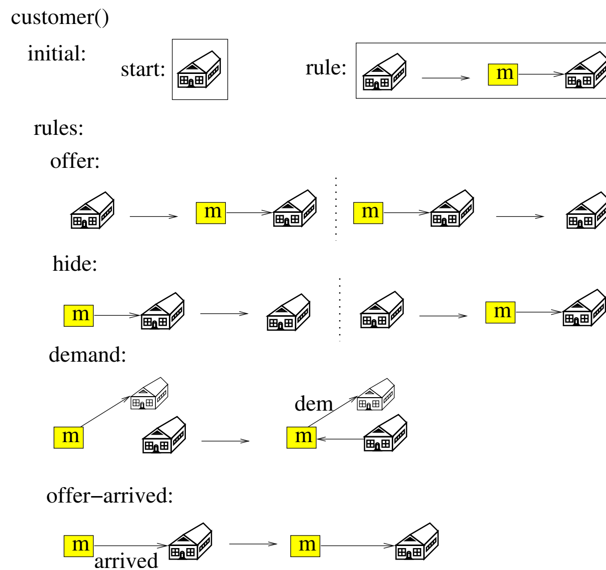

Fig. 11. The unit *customer*

The autonomous unit *customer* is depicted in Fig 11. It may offer and demand packages and in its private state it stores private packages that are not offered to the community. If a customer wants to offer a private package, it applies the product rule *offer* that inserts it into the common environment. On the other hand, it can hide offered packages with the rule *hide*. With the rule *demand* the customer demands a package p that is offered by another customer A. This is modeled by inserting a new edge from the customer to p and labeling the edge from A to p with *dem* representing in this way the fact that p cannot be demanded anymore. Finally, with the rule *offer-arrived*, the customer can convert a recently obtained package into an offered one.

The community for the basic operations of the pickup and delivery problem can now be defined as $pdp = (ini, \{truck, package, customer\}, goal)$ where *ini* is the grammar of Fig. 4, and the goal could be specified in this first approach such that all environments are accepted.

6 Semantics

In this section we describe the semantics of communities. In [4] a sequential semantics is given, but it is not fully adequate for the pickup and delivery problem because several trucks may move simultaneously and several packages may be loaded and reloaded at the same time. Hence, we adopt the parallel semantics introduced in [9]. But since private states and used transformation units were not considered in [9], we have to generalize the parallel semantics.

The operational semantics of communities consists of a set of transformation processes which are sequences of states where a *state* is a tuple $(ce, ps_1, \ldots, ps_k)$ of graphs such that ce is the current common environment and $ps_1 \ldots, ps_k$ are the current private states occuring in the community. A state transformation transforms one state into another by applying product rules of autonomous units in parallel. More precisely, let COM be a community consisting of a set AUT of autonomous units, a graph class expression *ini* specifying all possible initial environments and a common goal *goal*. Let ce be a graph specified by *ini*. Let $v_1, \ldots, v_k$ be the nodes of ce the behavior of which is modeled by the autonomous units $type(v_1), \ldots, type(v_k)$, respectively. Moreover, let $ps_1, \ldots, ps_k$ be graphs such that for $j = 1, \ldots, k$, ps_j is a private initial state of $type(v_j)$. Then the tuple $(ce, ps_1, \ldots, ps_k)$ is an *initial state* of COM. A sequence $s = (M_0, M_1, \ldots)$ of states with $M_i = (ce_i, ps_{i,1}, \ldots, ps_{i,k})$ is a *transformation process* of COM if

1. M_0 is an initial state of COM,
2. for $i = 1, \ldots, n$ if $s = (M_0, \ldots, M_n)$ and for $i \in \mathbb{N}$ if s is infinite, M_{i+1} is obtained from M_i as follows: There are $r_1, \ldots, r_k$ such that for $j = 1, \ldots, k$, r_j is a (parallel) product rule of $type(v_j)$, or a product rule of some used transformation unit of $type(v_j)$, or the empty product rule the application of which has no effect, such that
   - ce_{i+1} is obtained from ce_i by applying the common parts of $r_1, \ldots, r_k$ in parallel so that the local nodes are matched as required (see Section 4);[1]
   - for $j = 1, \ldots, k$ the graph $ps_{i+1,j}$ is obtained from $ps_{i,j}$ by applying the private part of r_j so that the local nodes are matched as required;
3. for $j = 1, \ldots, k$ the sequence $((ce_0, ps_{0,j}), (ce_1, ps_{1,j}), \ldots)$ is allowed by the *flattened*[2] control condition of $type(v_j)$.

Please note that the semantics of control conditions introduced in Section 2 must be generalized here to product rules, i.e. every control condition specifies

[1] In general, for applying rules in parallel, certain independence criteria must be satisfied (see e.g. [9]).

[2] We require that every autonomous unit can be flattened without changing its semantics (see also [3]).

sequences of pairs of graphs. This generalization can be done for the considered control conditions in a straightforward way. Moreover, the priority control conditions as used in this paper can be flattened as follows. Let *aut* be a unit with $(N, \leq_{aut})$ as control condition, i.e. N is composed of rules and used units of *aut*. Clearly, if N consists of rules only, its flattened condition $(\mathit{flat}(aut), \mathit{flat}(\leq_{aut}))$ is equal to $(N, \leq_{aut})$. Otherwise, for every used unit $t \in N$ with control condition $(N_t, \leq_t)$ let its flattened condition $(\mathit{flat}(t), \mathit{flat}(\leq_t))$ be already defined; and for every rule r in N, let $\mathit{flat}(r) = \{r\}$ and $\mathit{flat}(\leq_r) = \emptyset$. Then the flattened control condition of *aut* is equal to $(\mathit{flat}(aut), \mathit{flat}(\leq_{aut}))$ where $\mathit{flat}(aut) = \uplus_{n\in N}\mathit{flat}(n)$[3] and $\mathit{flat}(\leq_{aut})$ is the reflexive and transitive closure of

$$\bigcup_{n\in N} \mathit{flat}(\leq_n) \cup \{r_1 \leq r_2 \mid r_1 \in \mathit{flat}(i), r_2 \in \mathit{flat}(j),\ i \leq_{aut} j,\ i,j \in N\}.$$

This means that the rule set $\mathit{flat}(aut)$ of the flattened condition of *aut* consists of all rules occurring in N and in the flattened conditions of the used units in N. The priority relation consists of the priority relation between the rules in the flattened conditions of the used units. Additionally, for two rules r_1 and r_2 in $\mathit{flat}(aut)$ we have that r_1 is of a higher priority than r_2, if $t_1 >_{aut} t_2$ in the control condition of *aut* where for $i \in \{1, 2\}$, t_i is either equal to the rule r_i or t_i is a used unit and r_i is a rule of the flattened condition of t_i.

Every finite transformation process is *successful* if its last state is specified by the goal of the community. Every infinite transformation process is *successful* if it contains infinitely many states that satisfy the goal (see [9] for more details).

The formal framework of communities of autonomous units based on graph transformation does not only allow one to model interacting logistic processes, but provides also means for their analysis.

One important aspect is the possibility of correctness proofs which are usually done by induction on the lengths of derivation sequences. With respect to our case study, many properties which one would expect of a solution of the pickup and delivery problem can be verified. The following observation lists a few explicit examples of such properties.

Observation 1. For every state in the operational semantics of the community *pdp* the following holds.

- The current load of every truck is equal to the sum of the weights of all packages in the truck.
- The maximal capacity of every truck is not smaller than its current load.
- A truck only moves (i.e. the move unit is only applicable) if there are no incoming messages left.
- A package can only enter into a truck if both are at the same location.
- A package is never in two trucks.
- A package can only get out of a truck if the truck has reached the customer who demanded the package.

[3] $\uplus$ denotes the disjoint union.

– Every package cannot be demanded by more than one customer at the same time.

The proof is omitted because it is beyond the scope of this paper.

Another matter is the parallelism analysis. There is some machinery available in the area of graph transformation (see e.g. [5,15]) to find out which rules can be applied in parallel. This is very helpful with respect to any case study, because our semantics embodies parallelism explicitly. Unfortunately, there is not enough space for a more detailed consideration.

7 Related Work

In the literature there are some approaches that focus on modeling multi-agent or agent-oriented systems based on graph transformation. These approaches are closely related to our approach because of the special features inherent to agents such as autonomy or reactivity (cf. [16] where autonomous units are related to the VSK model of multi-agent systems, see e.g., [17]).

In [18] an approach for modeling agent-oriented systems is proposed that is based on UML and typed graph transformation. It concerns mainly the modeling process which consists of three stages (requirement specification, analysis, and design) where the second and the last stages are refinements of their predecessors. The relations between the distinct stages are formalized using typed graph transformation systems and graph processes. In the last stage, every agent corresponds to an active class where operations are modeled as graph transformation rules and the control component as a state chart.

In [19] an approach to model and verify multi-agent systems is given that is also based on typed graph transformation and UML. A complete system is composed of communities that can be entered or left by agents. A community is obtained by associating a culture specification with an environment specification where the former specifies social components such as roles and intentions and the latter specifies (physical) entities, agents as well as sensors and effectors. The whole system can be formalized as a graph transformation system.

Communities of autonomous units are also closely related to [20] where distributed systems are modeled by graph grammars that modify distributed graphs via distributed graph productions. Distributed graphs are network graphs with local graphs as node labels and graph morphisms as edge labels.

All three approaches are based on particular graph transformation approaches (single- and double-pushout) while our framework is independent of a particular graph transformation approach. Similarly, we employ a quite generic concept of control conditions while the other three approaches use particular control concepts or none at all. Moreover, in [18] and [19] certain types of multi-agent systems are formalized by graph transformation while autonomous units can be considered as an operational model of an axiomatic notion of multi-agent systems.

8 Conclusion

In this paper we have demonstrated that the basic operations of the pickup and delivery problem can be visually modeled in a rule-based way by means of a community so that central control is omitted, but spread over a set of autonomous units each of which specifies the behavior of a component occurring in the pickup and delivery problem, such as trucks, customers, and packages. The autonomous units communicate and interact in a common environment consisting of roads, customers, trucks, and packages and the actions of a unit comprise the controlled application of parallel product rules which modify the common environment of the community and the private state of the unit simultaneously and in a controlled way. Moreover, in order to keep large rule sets manageable, they can be divided into smaller transformation units. Semantically, a community specifies possibly infinite sequences of states consisting of the current common environment and the current private states of the units.

The presented case study points out that the private states and the use of product rules constitute an adequate and useful generalization of the hitherto defined autonomous units with parallel semantics [9]. Moreover, the case-study stresses that operations of logistic processes can be visually and easily modeled by graph transformation-based autonomous units, i.e. these operations which include message sending, moving around the environment, entering or leaving other units can be visually represented by means of small graph transformation rules.

In order to be able to present this case study within the scope of this paper we have simplified it w.r.t. various aspects. In an extended study we will investigate how the following aspects can be solved in a graph-transformational way. (1) A more detailed communication concerning prices, tours, etc. between the different units; (2) routing algorithms for the truck units; (3) capability of packages to change trucks; and (4) different behaviors of units of the same type.

Acknowledgement. We are very grateful to the anomymous reviewers of this paper for their helpful comments.

References

1. Hölscher, K., Klempien-Hinrichs, R., Knirsch, P., Kreowski, H.J., Kuske, S.: Autonomous units: Basic concepts and semantic foundation. In: [10], pp. 103–120
2. Klempien-Hinrichs, R., Kreowski, H.J., Kuske, S.: Typing of graph transformation units. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 112–127. Springer, Heidelberg (2004)
3. Kreowski, H.J., Kuske, S.: Graph transformation units with interleaving semantics. Formal Aspects of Computing 11(6), 690–723 (1999)
4. Hölscher, K., Kreowski, H.J., Kuske, S.: Autonomous units and their semantics—the sequential case. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 245–259. Springer, Heidelberg (2006)

5. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformation. Foundations, vol. 1. World Scientific, Singapore (1997)
6. Savelsbergh, M., Sol, M.: The general pickup and delivery problem. Transportation Science 29(1), 17–29 (1995)
7. Nagy, G., Salhi, S.: Heuristic algorithms for single and multiple depot vehicle routing problems with pickup and deliveries. European Journal of Operational Research 162(1), 126–141 (2005)
8. Fabri, A., Recht, P.: On dynamic pickup and delivery vehicle routing with several time windows and waiting times. Transportation Research Part B: Methodological 40(4), 335–350 (2006)
9. Kreowski, H.J., Kuske, S.: Autonomous units and their semantics - the parallel case. In: Fiadeiro, J., Schobbens, P. (eds.) Recent Trends in Algebraic Development Techniques, 18th International Workshop, WADT 2006. LNCS, vol. 4408, pp. 56–73. Springer, Heidelberg (2007)
10. Hülsmann, M., Windt, K. (eds.): Understanding Autonomous Cooperation & Control in Logistics The Impact on Management, Information and Communication and Material Flow. Springer, Heidelberg (2007)
11. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundamenta Informaticae 26(3,4), 287–313 (1996)
12. Klempien-Hinrichs, R., Kreowski, H.J., Kuske, S.: Rule-based transformation of graphs and the product type. In: van Bommel, P. (ed.) Transformation of Knowledge, Information, and Data: Theory and Applications, pp. 29–51. Idea Group Publishing, Hershey (2005)
13. Schürr, A.: Specification of graph translators with triple graph grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 151–163. Springer, Heidelberg (1995)
14. Klempien-Hinrichs, R., Knirsch, P., Kuske, S.: Modeling the pickup-and-delivery problem with structured graph transformation. In: Proc. APPLIGRAPH Workshop on Applied Graph Transformation, pp. 119–130 (2002)
15. Ehrig, H., Kreowski, H.J., Montanari, U., Rozenberg, G. (eds.): Handbook of Graph Grammars and Computing by Graph Transformation. Concurrency, Parallelism, and Distribution, vol. 3. World Scientific, Singapore (1999)
16. Timm, I.J., Knirsch, P., Kreowski, H.J., Timm-Giel, A.: Autonomy in software systems. In: [10], pp. 255–273
17. Wooldridge, M., Lomuscio, A.: A logic of visibility, perception, and knowledge: Completeness and correspondence results. In: Proc. Third International Conference on Pure and Applied Practical Reasoning, London, UK (2000)
18. Depke, R., Heckel, R., Küster, J.M.: Formal agent-oriented modeling with UML and graph transformation. Science of Computer Programming 44, 229–252 (2002)
19. Giese, H., Klein, F.: Systematic verification of multi-agent systems based on rigorous executable specifications. International Journal on Agent-Oriented Software Engineering (IJAOSE) 1(1), 28–62 (2007)
20. Taentzer, G.: Parallel and Distributed Graph Transformation: Formal Description and Application to Communication-Based Systems. PhD thesis, TU Berlin. Shaker Verlag (1996)