Modeling Agent Systems with Distributed Transformation Units

Sabine Kuske\textsuperscript{1} Peter Knirsch\textsuperscript{2}

\textit{Fachbereich 3}  
\textit{Universität Bremen}  
\textit{Bremen, Germany}

Abstract
Agent systems have become more and more important in computer science. They allow to implement complex distributed systems composed of communicating autonomous entities. Transformation units constitute a structuring principle for graph transformation systems which split up large sets of rules, but still graphs are transformed as a whole. Recently, distributed transformation units have been introduced as an extension of transformation units to distributed graphs and distributed graph transformation. In this paper it is illustrated how different features of agent systems can be smoothly modeled in a uniform way by distributed graph transformation systems. For this purpose an agent system case study with simple agents communicating via blackboards and message passing is presented.

1 Introduction
Agents and agent systems\textsuperscript{[14]} can be seen as a new programming paradigm suitable for the new challenges of a distributed and heterogeneous complex information system structure as it is imposed e.g. by the internet. A vast amount of different information sources and sites make structuring inevitable. Therefore, the distribution structure as well as the different functionalities have to be adequately reflected in new software architectures. Agents constitute a means for functional and data abstraction, as objects do, together with an environment they are located in. But different from object orientation, the focus lies more on autonomy and cooperation than on typing. This enables agents to be more flexible and hence software systems to be more fault tolerant. But this advantage does not come for free. The semantics of distributed

\textsuperscript{1} Email: kuske@informatik.uni-bremen.de  
\textsuperscript{2} Email: knirsch@informatik.uni-bremen.de
concurrent systems, capable of somewhat intelligent and autonomous actions, is hard to define.

In this paper we propose to model agent systems with distributed transformation units which are an extension of transformation units presented in e.g. [10,9]. Transformation units are graph-and-rule-centered i.e. they allow to visualize system states by graphs and state transformations by the application of graph transformation rules. In the literature one can find an amount of examples which prove the usefulness of graph transformation to model or to support the modelization of complex (software) systems (cf. [6]). Moreover, transformation units constitute a structuring concept for graph transformation systems that on the one hand allows to group large sets of graph transformation rules into small and manageable parts. On the other hand, transformation units provide control conditions in order to regulate the transformation process. Distributed transformation units extend transformation units in the sense that distributed states are transformed concurrently by a set of local transformation units.

The presented paper is based on the idea that agents can be modeled as local transformation units and the specific data (static knowledge) of each agent as a local part of a distributed graph. Agents can modify their environment, i.e. the distributed graph, concurrently provided that transformation of shared data is synchronized in some way. The behavior of every agent is regulated by the control condition of the transformation unit representing it. Moreover, communication between agents is performed via the transformation of interfaces, i.e. shared data. A first approach regarding agents as transformation units having local rules and units is given in [7]. The differences of the work in [7] are that (1) agents do not operate on a distributed graph, i.e. all agents have the same environment; (2) the semantics of agent system is interleaving basing on simple parallel rule applications, i.e. agents do not operate concurrently. An extension to concurrent agents is only shortly sketched.

This paper is organized as follows. The next section gives a brief overview of agent systems. Section 3 shortly presents distributed transformation units and illustrates the basic concepts with some examples. These examples are extended to the case study of finding shortest paths in a cooperative way in Section 4. It turns out that different kinds of communication between agents—such as message passing and blackboard communication—as well as basic features of agents—such as autonomy, cooperation, and reactivity—can be smoothly modeled in a straightforward way using just one uniform framework, i.e distributed graph transformation. To our knowledge, the only other extensive approach to the modelization of agent systems using graph transformation is given by Depke, Heckel, and Küster in e.g. [4] or [3]. In Sect. 5 their approach is shortly described and compared with ours. Some concluding remarks are given in the last section.
2 Agents and agent systems

Agent systems originally where invented in cognition science. Marvin Minsky stated in his well-known book “The Society of Mind” [12] that the human brain is structured in autonomous parts. Human intelligence emerges from the cooperation of these parts. The idea of researches from the Artificial Intelligence then was to simulate this cooperation inside the brain and equip software systems and robots with human-like intelligence [11]. But to this respect Artificial Intelligence failed. While investigating complex distributed scenarios, it turned out that modeling these systems by means of agents is a well suited technique for software engineering to design distributed reactive and open systems. As objects in the object oriented modeling of monolithic systems, agents are a quite natural abstraction of complex distributed and open systems. Agents are special objects, cf. active objects also part of the specification of the UML [2], which are in contrast to standard objects autonomous, i.e. not controlled by anyone else. They sense their environment they are situated in and react accordingly. Although there are a lot of different approaches to agent architecture most kinds have similar concepts. Agents comprise different parts. They have a local representation of their static knowledge and sets of rules to manipulate local data and the external environment that is shared by different agents. To enable communication agents need do obey certain demands. For instance, they need to have some public ports, or addresses where others can send messages to. These ports should not be removable by others. Local parts can only be changed by the respective agent. The common parts can only be manipulated in a defined way, to guarantee that the information stay accessible for the other agents. As a very outstanding feature agents are supposed to be autonomous, e.g. an agent decides if it returns a message on a request or not. In contrast a method of an object that is called is always supposed to execute. It is also very important that agents are reactive. They keep sensing and may react if a certain situation occurs. One way to react, and again an important agent feature, is communication. As communication is a very important issue especially concerning cooperation and resolving of conflicts, it is useful to study different kinds of communication. In the approach given here we use message passing, i.e. agents send messages to each other and also blackboard communication, i.e. agents change common data to exchange information. Other more sophisticated communication mechanisms can be simulated using the basic ones. Especially the broadcasting of messages to all known agents and the often used contract net protocol [13], i.e. negotiation based task allocation, can be implemented using these communication primitives.

An agent system then is simply a set of agents situated in and changing a common environment. In the following section we introduce a data structure, a distributed graph, as well as the transformation of distributed graphs as a basis for our approach to modeling agents and agent systems.
3 Distributed transformation units

Distributed transformation units transform distributed graphs concurrently. For reasons of space limitations a simplified and partly informal introduction to distributed units is given in this section. For a detailed and formal description the reader is referred to [8].

Every distributed graph and every distributed unit is defined over a distribution structure which determines how many local parts compose a distributed system and which of them may share information or manipulate common data.

**Definition 3.1 [Distribution structure]** A distribution structure \( DS \) consists of a set \( LocV \) of local nodes, a set \( IntV \) of interface nodes, and a set \( Att \subseteq LocV \times IntV \) of edges attaching local nodes with interface nodes. A distribution structure with exactly one local node to which every interface node is attached, is called a local distribution structure. Every local node \( v \) of \( DS \) induces a local distribution structure \( ind(DS, v) \) consisting of \( v \) and all interface nodes attached to \( v \).

In the following we assume an arbitrary but fixed distribution structure \( DS = (LocV, IntV, Att) \). The distributed structure used in our running example is shown in the middle layer of Fig. 1 where \( loc_1, loc_2, \) and \( loc_3 \) are local nodes and \( int_1 \) and \( int_2 \) are interface nodes.

A distributed graph over \( DS \) consists mainly of a local graph for every local node and an interface graph for every interface node. Local graphs are graphs that represent local data or local environments of agents and interface graphs are graphs which represent shared information.

**Definition 3.2 [Distributed graph]** A distributed graph over \( DS \) is a mapping graph which associates a graph with every node of \( DS \).

In Fig. 1 it is shown that each local node \( loc_i \) is connected to a local graph \( lg_i \) and each interface node \( int_i \) is connected to an interface graph \( ig_i \). Instances of local and interface graphs are given in Fig. 2. In the example that follows we have three different local graphs not in the intersection of the gray rectangles and two different interface graphs. One interface contains the graph within the intersection of the three rectangles. The second just contains the node labeled \( A_2 \).
A distributed transformation unit is composed of a set of local units and a set of interface units. Both local units and interface units encapsulate sets of rules. (In this simplified version of distributed transformation units, a rule can be a graph transformation rule or an imported transformation unit as introduced in [9].) The rules of local units are used to transform the local environment of the local units. The rules of interface units can be used by local units to transform shared parts of a distributed graph. For this purpose, local units contain synchronization rules that prescribe which rules of the interface units have to be applied in parallel with the rules of the local units. It is possible that several local units transform an interface graph in parallel. A schema of the distributed transformation unit used in our running example is shown in the bottom of Fig. 1 where $lu_1$ and $lu_2$ are local units and $iu$ is an interface unit.

Apart from the described components, local and interface units may contain initial and terminal graph class expressions that specify sets of graphs. Moreover, every interface unit contains a control condition to regulate its graph transformation process. We assume that the components of local and interface units together with their semantics are given by some underlying graph transformation approach (cf. [1,9]).

Assumption.

In the following we assume the existence of a class $G$ of graphs, a class $R$ of rules, a class $E$ of graph class expressions, and a class $C$ of control conditions, with $SEM(R) \subseteq G \times G$, $SEM(e) \subseteq G$, and $SEM(c) \subseteq G \times G$ for every $R \subseteq R$, $e \in E$, and $c \in C$.

Local transformation units are defined over local distribution structures.

**Definition 3.3** [Local unit] A local unit $lu$ over a local distribution structure $DS$ consists of an initial graph class expression $I \in E$, a terminal graph class
expression $T \in \mathcal{E}$, a control condition $C \in \mathcal{C}$, and a set $S$ of synchronization rules, i.e. a set of a partial mappings from the nodes of $DS$ to $\mathcal{R}$.

Interface units are defined as follows.

**Definition 3.4** [Interface unit] An interface unit $iu$ consists of an initial graph class expression $I \in \mathcal{E}$, a terminal graph class expression $T \in \mathcal{E}$, a control condition $C \in \mathcal{C}$, and a set $R \subseteq \mathcal{R}$ of rules.

A distributed transformation unit over $DS$ consists of a local transformation unit for every local node of $DS$ and an interface unit for every interface node.

**Definition 3.5** [Distributed transformation unit] A distributed transformation unit over $DS$ is a mapping $dtu$ (depicted in Fig. 1) which assigns an interface unit to every interface node of $DS$, and a local unit over $ind(DS, v)$ to every local node $v$ of $DS$ such that every rule assigned to an interface node $i$ by some synchronization rule of $dtu(v)$ is contained in the interface unit $dtu(i)$.

The basic operation of distributed transformation units are distributed transformation steps, in which local units transform local graphs with interfaces concurrently by applying synchronization rules.

**Definition 3.6** [Distributed transformation step] Let $dtu$ be a distributed transformation unit over $DS$. Let $l_1, \ldots, l_n \in LocV$ and for $i = 1, \ldots, n$, let $s_i$ be a synchronization rule of $dtu(l_i)$. Then a distributed graph $graph$ over $DS$ is transformed into a distributed graph $graph'$ over $DS$ via a distributed transformation step if for every node $v$ of $DS$ the following holds.

- if $v = l_i$ for some $i \in \{1, \ldots, n\}$ and $s_i(l_i)$ is defined, $graph(l_i)$ is transformed into $graph'(l_i)$ by applying $s_i(l_i)$, i.e. $(graph(l_i), graph'(l_i)) \in SEM(\{s_i(l_i)\})$;
- if $v \in IntV$, $graph(v)$ is transformed into $graph'(v)$ by the parallel application of all rules in $\mathcal{R}$ where $\mathcal{R}$ is the set of rules assigned to $v$ by some synchronization rule in $\{s_1, \ldots, s_n\}$, i.e. $(graph(v), graph'(v)) \in SEM(\mathcal{R})$;
- otherwise, $graph(v) = graph'(v)$.

Semantically, a distributed transformation unit transforms a distributed graph by applying a sequence of distributed transformation steps. Moreover, the control conditions and the graph class expressions of the distributed unit must be satisfied.

**Definition 3.7** [Distributed semantics] Let $dtu$ be a distributed transformation unit over $DS$. Then the distributed semantics $DIST(dtu)$ is a binary relation on distributed graphs over $DS$ where $(graph, graph')$ is in $DIST(dtu)$ if (1) $graph'$ is obtained from $graph$ by a sequence of distributed transformation steps, (2) for all $v$ in $DS$ $graph(v)$ is specified by the initial graph class expression of $dtu(v)$, $graph'(v)$ is specified by the terminal graph class expres-
sion of \( dt(u(v)) \), and \((graph(v), graph'(v))\) is specified by the control condition of \( dt(u(v)) \).

4 Example

The example illustrated here is of the domain of distributed problem solving of cooperative agents. All agents on their own are not able to comply a task. This is either because of incomplete knowledge or of missing capabilities. But cooperating it can be done. The running example is taken from the travel domain but can easily be extended to other logistic supply chains. A user which is represented by the user agent \( A_u \) wants some information on the travel from location \( a \) to location \( b \). Therefore, it sends a message to the travel agent \( A_1 \). As one can see from Fig. 2 the different locations and the agents are represented as nodes of the distributed graph. To send messages to agents, agents and messages must be part of the (public) interface graph. Hence, a rule for sending messages is part of the interface unit.

The interface rules

\[ rcv \xrightarrow{ } rcv \xrightarrow{snds,t} \text{ and } rcv \xrightarrow{snds,t} \xrightarrow{ } rcv \]

implement asynchronous message passing from a sender \( snd \) to a receiver \( rcv \) and removing by graph transformation. The content of the message is the name of the sender \( snd \) and the source \( s \) and target \( t \) of the travel. If we do not want that the received messages are non-deterministically selected, we easily could implement queues of messages. Using the control conditions the broadcasting of messages to all agents represented in the interface graph can be modeled. The sending of a message from the user agent to agent \( A_1 \) is shown in Fig. 3.

Additionally we have two more interface rules to add or delete labeled edges between the nodes in the interface, i.e.

\[ 1 \xrightarrow{x} 2 \text{ and } 1 \xrightarrow{x} 2 \rightarrow 1 \xrightarrow{} 2 \]
If an agent receives a message it may or may not react. This choice reflects reactivity and autonomy. For every kind of reaction it needs to have local rules or synchronization rules. Both travel agents have the same local rules. The following rule is to extract the source and the destination of the travel from the message sent and to mark both locations in the local graph. It has to be applied synchronously with the interface rule above which removes the message $A_u, a, b$ attached to $A$ in the interface in order to prevent a second processing.

$$l \mathrel{\rightarrow} l mA, s A, t$$

The application of the corresponding synchronization rule is shown in Fig. 4.

To make sure that there is a path from the source to the destination and that it is optimal, the agent executes a shortest path algorithm given by its local rules:

$$x \leq g + h \quad \text{and} \quad g + h \rightarrow \min(g, h).$$

The first rule adds new paths as long as there are no shorter ones. The dashed arc is meant to be a negative application condition symbolizing a forbidden context. The second rule deletes an edge between two nodes if there is a shorter one between the same nodes. The agent applies those two rules in arbitrary order as long as it is possible. After the termination we have the minimal distance from the source to the destination on the edge in between if it exists.\(^3\) This is checked by a further local rule:

$$A_{x, s} \quad A_{x, t} \quad A_{x, s} \quad A_{x, t}$$

This rule does not change anything but is used to trigger the sending of a message in a synchronization rule. If agent $A_1$ does not have an edge between source and destination node, it does not provide this route and is forced to write a message to the second travel agent $A_2$ which is not in the same interface.

\(^3\) In [5] it is shown how the shortest-path algorithm of Floyd and Warshall can be implemented with transformation units by using the two described rules.
as the user asking for help using

\[
\begin{array}{c}
A_2 \\
\rightarrow \\
A_1, k, m
\end{array}
\]

Here an arbitrary node \( k \) as source is selected. This behavior shows that communication and cooperation is important when having different resources and capabilities. Because agent \( A_2 \) has the same capabilities as agent \( A_1 \) the processing of the request works analogously. If the second agent can provide a route from \( k \) to the target it writes it to the interface and deletes the request nodes. Agent \( A_1 \) senses the change of the interface copies the edge in its local part and starts over calculating the shortest path again. Afterwards it writes the demanded route from \( a \) to \( b \) in the interface graph and deletes its local request nodes. The user agent is free to take this offer. Here adding edges to the interface graph corresponds to blackboard communication, because shared data, i.e. the interface graph, is used to communicate without explicitly addressing a sender or receiver.

Usually, the task of finding cooperative partners would be accomplished by using the contract net protocol. An initiator who is in need of a certain service sends request messages to all possible partners and then waits for offers and starts a negotiation. Here, due to simplicity reasons we use a less well structured version. But it is easy to see that a blackboard reflecting the current state of the negotiation could be used to simulate this kind of communication.

5 Another graph transformation based approach

As already stated in the introduction, to our knowledge the only other approach which models agent-oriented systems with graph transformation was proposed by Depke, Heckel and Küster in e.g. [4] or [3]. In contrast to our approach, they use UML modeling techniques and focus on the development process of agent-oriented systems starting from a requirement specification (expressed by a use case diagram) and ending with the design phase. More precisely, on the analysis level as well as on the design level an agent-oriented system is split into a structural model, a dynamic model, and a functional model. The structural models are represented by class diagrams containing active objects for agents, which are provided with a particular compartment for messages on the analysis level. In the design phase a compartment for signatures of agent operations is added. The dynamic model of the analysis phase consists of a sequence diagram that specifies communication between agents. In the design phase the dynamic model comprises a kind of state machine prescribing possible orderings in which an agent may apply its operations. The functional model of the analysis phase consists of global graph transformation rules which specify pre- and post-conditions of system states with respect to specific scenarios of the use case diagram. On the design level the functional model consists of graph transformation rules which implement the operations an agent can perform. The semantic consistency between the
different models is formalized by using graph transformation theory.

Summarizing, in [3] an agent is an active object the operations of which are specified with (typed) graph transformation rules and their application order with some kind of state machine. Communication between agents is specified by sequence diagrams and pre- and post-conditions of system states by graph transformation rules. Consistency of the different concepts can be formalized using graph transformation theory.

Two basic common features of both approaches is that the operations an agent performs are applications of graph transformation rules and that the operational semantics of agents is based on derivations, i.e. sequences of applications of graph transformation rules. Some main differences of both approaches are listed in the following table. The formal relation between them should be worked out in the future.

<table>
<thead>
<tr>
<th></th>
<th>Approach of [3]</th>
<th>Approach with distributed transformation units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>Active object of a class diagram</td>
<td>Local unit</td>
</tr>
<tr>
<td>System state</td>
<td>Object diagram</td>
<td>Distributed graph</td>
</tr>
<tr>
<td>Description of the combined effect of an interaction</td>
<td>Global graph transformation rules</td>
<td>Semantics of distributed graph transformation steps</td>
</tr>
<tr>
<td>Communication between agents</td>
<td>Sequence diagram</td>
<td>Reading from and writing to interface graphs</td>
</tr>
<tr>
<td>Control</td>
<td>State machine</td>
<td>Control condition</td>
</tr>
<tr>
<td>Environment</td>
<td>Reachable items in the object diagram</td>
<td>Local graph with interfaces</td>
</tr>
<tr>
<td>Semantic aspects</td>
<td>Semantic consistency of models is formalized based on typed graph transformation</td>
<td>Semantics of an agent system is a binary relation on distributed graphs based on distributed graph transformation</td>
</tr>
</tbody>
</table>
6 Conclusion

In this paper it has been illustrated how distributed graph transformation units can be applied in the area of agent systems. This has been shown with a small case study of two travel agents and a user agent. The travel agents had to find out an optimal tour from a source to a destination requested by a user agent. For this purpose they have to cooperate if one agent provides only parts of the requested tour. Otherwise they can work independently of each other and concurrently. We believe that the presented case study can be generalized in a straight-forward way to the case of an arbitrary number of travel agents and user agents. Although it is rather simple and small, the case study underlines that distributed transformation units are suitable to

- visualize the environments of agent systems and the agents’ dependencies;
- implement two fundamental communication techniques of agent systems, namely message passing and blackboard communication;
- keep typical features of agents, like reactivity, cooperation, and autonomy;
- constitute a uniform approach to specify all layers of an agent architecture, e.g. structuring, knowledge representation, communication, using distributed graph transformation;
- give a formal semantics to agent systems.

In the future we will systematically investigate how agent systems can be formalized based on graph transformation and compare our work with other approaches such as [3] and those from the Artificial Intelligence community to formalize agent systems some of them presented in [15,14]. Moreover, it should be investigated how our concepts can incorporate more dynamic aspects such as the generation and termination of agents and the transformation of distribution structures.

Acknowledgement

We would like to thank the referees for their valuable comments.

References


