Integrating Free-Flow Architectures with Role Models Based on Statecharts

Danny Weyns, Elke Steegmans, and Tom Holvoet

AgentWise, DistriNet
Department of Computer Science K.U.Leuven
Celestijnenlaan 200 A
B-3001 Leuven, Belgium
{Danny.Weyns,Elke.Steegmans,Tom.Holvoet}@cs.kuleuven.ac.be

Abstract. Engineering non-trivial open multi-agent systems is a challenging task. Our research focusses on situated multi-agent systems, i.e. systems in which agents are explicitly placed in a context – an environment – which agents can perceive and in which they can act. Two concerns are essential in developing such open systems. First, the agents must be adaptive in order to exhibit suitable behavior in changing circumstances of the system: new agents may join the system, others may leave, the environment may change, e.g. its topology or its characteristics such as throughput and visibility. A well-known family of agent architectures for adaptive behavior are free-flow architectures. However, building a free-flow architecture based on an analysis of the problem domain is a quasi-impossible job for non-trivial agents. Second, multi-agent systems developers as software engineers require suitable abstractions for describing and structuring agent behavior. The abstraction of a role obviously is essential in this respect. Earlier, we proposed statecharts as a formalism to describe roles. Although this allows application developers to describe roles comfortably, the formalism supports rigid behavior only, and hampers adaptive behavior in changing environments.
In this paper we describe how a synergy can be reached between free-flow architectures and statechart models in order to combine the best of both worlds: adaptivity and suitable abstractions. We illustrate the result through a case study on controlling a collection of automated guided vehicles (AGVs), which is the subject of an industrial project.

1 Introduction

Dealing with the increasing complexity of developing, integrating and managing open distributed applications is a continuous challenge for software engineers. In the last fifteen years, multi-agent systems have been put forward as a key paradigm to tackle the complexity of open distributed applications. In this paper we focus on situated multi-agent systems[1](situated MASs) as a generic approach

[1] Alternative descriptions are behavior-based agents [4], adaptive autonomous agents [22] or hysteretic agents [16][14].

R. Choren et al. (Eds.): SELMAS 2004, LNCS 3390, pp. 104–120, 2005.

to develop self-managing open distributed applications. In particular, we propose an approach that combines aspects of adaptive agent architectures with ideas of rigid modeling of agent behavior for developing these kinds of multi-agent systems.

In situated MASs, agents and the environment constitute complementary parts of a multi-agent world which can mutually affect each other [33]. Situatedness places an agent in a context in which it is able to perceive its environment and in which it can (inter)act. Intelligence in a situated MAS originates from the interactions of the agents in their environment rather than from the capabilities of the individual agents. While interacting, agents form an organization in which they all play and execute their own role(s) in that organization.

The approach of situated MASs has a long history. R. Brooks [4][5] identified the key ideas of *situatedness*, *embodiment* and *emergence of intelligence*. L. Steels [31] and J. L. Deneubourg [11] introduced the basic mechanisms for agents to coordinate through the environment: *gradient fields* and *marks*. P. Maes [22] adopted the early robot-oriented principles of reactivity in a broader context of software MASs. J. Ferber and A. Drogoul [13], M. Dorigo [12], V. Parunak [27] and many other researchers drew inspiration from social insects and adopted the principles in situated MASs. Where the approach of situated MASs started from the rejection of classical agency based on symbolic AI, nowadays the original opposition tends to evolve towards convergence with different schools emphasizing different aspects. The researchers, although having different points of view, are very complementary, and each have their own applications.

Situated MASs have been applied with success in numerous practical applications over a broad range of domains, e.g. manufacturing scheduling [28], network support [3] or peer-to-peer systems [1]. The benefits of situated MASs are well known, the most striking being flexibility, robustness and efficiency.

During the last two years, we developed an agent architecture that enables advanced adaptive agent behavior. The architecture is a hierarchical free-flow architecture which integrates the concept of situated commitments. Situated commitments allow an agent to bias action selection towards actions in its commitments.

Besides the theoretical work on agent architectures, we have been confronted with application engineers who require software engineering support for developing concrete, real-world MASs, the applications include active networking, manufacturing control and supply chain networks. These software engineers require simple and comfortable modeling languages for functionally describing agent behavior. A modeling language based on statecharts resolved this requirement. However, a statechart specification of agent behavior is typically a static, rigid model in that it leaves little room for adaptive and explorative behavior. As a result, the agents in the applications performed behavior that was sometimes unable to adapt to different environmental situations.

Free-flow architectures allow adaptive behavior, yet it is unrealistic to assume that software engineers – starting from the analysis of the problem domain – build a complex free-flow architecture for complex applications, where agents

can perform many actions. For such applications, the architecture quickly becomes unmanageable. We aim to combine the best of both worlds, i.e. the best of adaptive architectures and simple modeling languages. To that end, we retain a flexible action selection mechanism, but complement its description with statecharts. Here, we refrain from considering a statechart description of agent behavior as a kind of sequence chart, but rather use statecharts to describe role composition and to structure related actions within roles only.

This paper is structured as follows. In section 2 we introduce free-flow architectures and give an overview of the statechart formalism we have developed. We discuss problems we encountered when applying them in practice. Section 3, the core of the paper, describes the combined adoption of free-flow architectures and the statechart modeling language. We illustrate our explanation with an example application. Section 4 discusses how the software engineering approach proposed in this paper relates to existing agent-oriented methodologies. Finally, in section 5 we conclude the paper.

2 Free-Flow Architectures and Statechart Models

In this section we start with a brief introduction of free-flow architectures. Then we give a short overview of the statechart formalism we have developed for modeling agent behavior. For both, we point out a number of problems we encountered when applying them in practice. Subsequently we outline an approach to combine free-flow architectures with statechart models.

2.1 Free-Flow Architecture for Adaptive Behavior

Open MASs are characterized by dynamism and change: new agents may join the system, others may leave, the environment may change, e.g. its topology or its characteristics such as throughput and visibility. To cope with such dynamism the agents must be able to adapt their behavior according to the changing circumstances. A well-known family of agent architectures for adaptive behavior are free-flow architectures.

Before we introduce free-flow architectures, we first clarify our perspective on adaptability in this paper. Here we look at adaptability as an agent's ability to deal with different kinds of situations in its environment in a flexible way. We do not look at adaptability in the sense of learning, i.e. as an agent's ability to adjust its behavior in certain kinds of situations over time, according to good or bad experiences of recent choices.

Free-flow architectures are first proposed by Rosenblatt and Payton in [29]. In his Ph.D thesis, T. Tyrrell [32] demonstrated that hierarchical free-flow architectures are superior to flat decision structures, especially in complex and dynamic environments. The results of Tyrrell's work have been very influential, for a recent discussion see [6].

An example of a free-flow architecture is depicted in Fig. 1.

The hierarchy is composed of *activity nodes* (in short nodes) which receive information from internal and external stimuli in the form of *activity*. The nodes

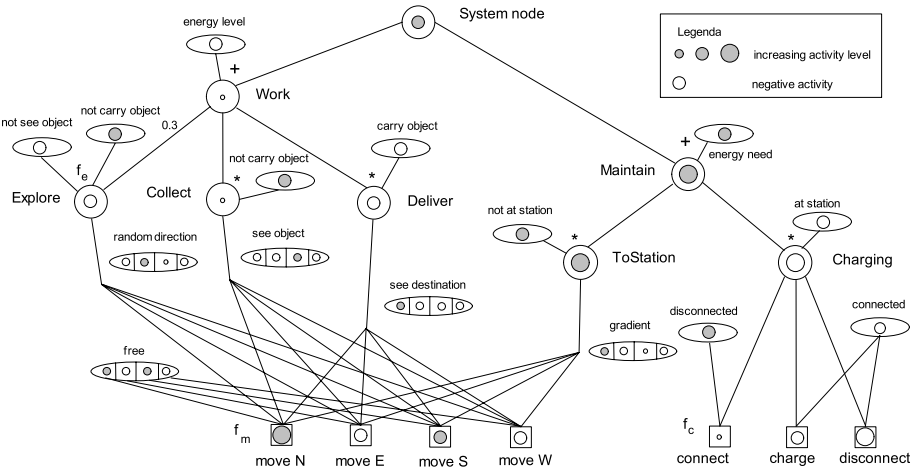

Fig. 1. An example of a free-flow architecture.

feed their activity down through the hierarchy until the activity arrives at the *action nodes* (i.e. the leaf nodes of the tree) where a winner-takes-it-all process decides which action is selected. The main advantages of free-flow architectures are:

- Stimuli can be added to the relevant nodes avoiding the "sensory bottleneck" problem. In a hierarchical decision structure, to make correct initial decisions, the top level has to process most of the sensory information relevant to the lower layers. A free-flow architecture does not "shut down" parts of the decision structure when selecting an action.
- Decisions are made only at the level of the action nodes; as such all information available to the agent is taken into account to select actions.
- Since all information is processed in parallel the agent can take different preferences into consideration simultaneously. E.g. consider an agent that moves to a spotted object but is faced with a neighboring threat. If the agent is only able to take into account one preference at a time it will move straight to the spotted object or move away from the threat. With a free-flow tree the agent avoids the threat *while* it keeps moving towards the desired object, i.e. the agent likely moves around the threat towards a spotted object.

Fig. 1 depicts a free-flow tree of the action selection of a simple robot. This robot lives in a grid world where it has to collect objects and bring them to a destination. The robot is supplied with a battery that provides energy to work. The robot has to maintain its battery, i.e. when the energy level of the battery falls below a critical value the robot has to recharge the battery at a charge station. The left part of the depicted tree represents the functionality for the robot to search, collect and deliver objects. On the right, functionality to maintain the battery is depicted. The *System node* feeds its activity to the *Work* node and the *Maintain* node. The *Work* node combines the received activity

with the activity from the *energy level* stimulus. The "+" symbol indicates that the received activity is summed up. The negative activity of the *energy level* stimulus indicates that little energy remains for the robot. As such the resulting activity in the *Work* node is almost zero. The *Maintain* node on the other hand combines the activity of the *System node* with the positive activity of the *energy need* stimulus, resulting in a strong positive activity. This activity is fed to the *ToStation* and the *Charging* nodes. The *ToStation* node combines the received activity with the activity level of the *not at station* stimulus (the "*" symbol indicates they are multiplied). In a similar way the *Charging* node combines the received activity with the activity level of the *at station* stimulus. This latter is a binary stimulus, i.e. when the robot is at the charge station its value is positive (true), otherwise it is negative (false). The *ToStation* node feeds its positive activity towards the action nodes it is connected with. Each moving direction receives an amount of activity proportional to the value of the *gradient* stimulus for that particular direction. *gradient* is a multi-directional stimulus. The value of this stimulus (for each direction) is based on the sensed value of the gradient field that is transmitted by the charge station. In a similar way, the *Charging* node and the child nodes of the *Work* node (*Explore*, *Collect* and *Deliver*) feed their activity to the action nodes they are connected with. Action nodes that receive activity from different nodes combine that activity according to a specific function. The action nodes for moving actions use a function f_m to calculate the final activity level. A possibility definition of this function is the following:

$$A_{moveD} = \max\left[(A_{Node} \ + \ A_{stimulusD}) \ * \ A_{freeD}\right]$$

Herein is A_{moveD} the activity collected by a move action, D denotes one of the four possible directions, i.e. $D \in \{N, E, S, W\}$. A_{Node} denotes the activity received from a node. The move actions are connected to four nodes: $Node \in \{Explore, Collect, Deliver, ToStation\}$. With each node a particular *stimulus* is associated. $stimulus \in \{random\ direction, see\ object, see\ destination, gradient\}$ are all multi-directional stimuli with a corresponding value for each moving direction. Finally, *free* is a multi-directional binary stimulus that indicates whether the way to a particular direction is free for the robot to move to.

When all action nodes have collected their activity the node with the highest activity level is selected for execution. In the example, the *ToStation* node is clearly dominant over the other nodes connected to actions nodes. Currently the East and West directions are blocked (see the *free* stimulus), leaving the robot two possibilities to move towards the charge station: via North or via South. According to the values of the guiding gradient field, the robot will move northwards, see Fig. 1.

For the simple robot in the example, the free-flow tree is already fairly complex. For a non-trivial agent however, the overall view of the tree quickly becomes very cluttered. When a change is made in one part of such a tree it becomes unclear how this affects the other parts. Although free-flow trees are at best developed with a focus on a particular functionality of the agent, the archi-

tecture itself does not support any structure. From our experiences we learned that it is unrealistic to assume that software engineers build a complex free-flow architecture for complex applications, where agents can perform many actions. For such applications, the architecture quickly becomes unmanageable, it is no longer possible to have an overall view of the architecture.

2.2 Statechart Models

To develop non-trivial open MASs software engineers require suitable abstractions for describing and structuring agent behavior. The abstraction of a role obviously is essential in this respect. Roles are quite general as core abstractions for designing MASs, see e.g. Gaia [9], MESSAGE [8] and also [15][25]. Similar to the definition in [35] we regard a role as an agent's functionality in the context of an organization. Roles provide the building blocks for the social organization of a MAS. Agents are linked to other agents by the roles they play in the organization. The links can be explicit, e.g. a set of agents that pass objects along a chain; or implicit, e.g. in an ant colony a dynamic balance exists between ants that supply the colony with food and ants that maintain the nest.

A number of researchers have proposed state-based approaches to model agent behavior. In SmartAgent [17], UML state machine models are used to model JADE behaviors. [24] points to the strength of statecharts as a constraint mechanism for agent interaction protocols. These and other related work use statecharts to model agent behavior with a focus on inter-agent communication. [18] and [2] are examples in which state machines are used to model reactive behavior. In previous work, we proposed statecharts as a formalism to describe agent behavior, see [19]. In that work we used a statechart formalism to model the behavior of situated agents in terms of roles, with a focus on reusing roles in different applications. Therefore, we extended the standard statecharts notation with new concepts, such as pre-action and post-action. Fig. 2 depicts an example of a role model for a scouting agent.

Although a statechart specification of agent behavior is simple to design and to understand, it is typically a static, rigid model in that it leaves little room for adaptive and explorative behavior. Practical experience with the statechart formalism brought up a number of considerations:

- Action sequences are defined statically. The designer has to enumerate all possible state transitions that can occur, or at least he has to distinguish between discrete categories of environmental situations and corresponding behavioral acts.
- The statechart formalism as developed is in principle only applicable for deterministic agent systems. MASs however, are typically non-deterministic. It is possible to integrate non-determinism in the modeling language, however this would complicate the models significantly. As a result, the agents in the applications performed behavior that was sometime unable to adapt to different environmental situations.

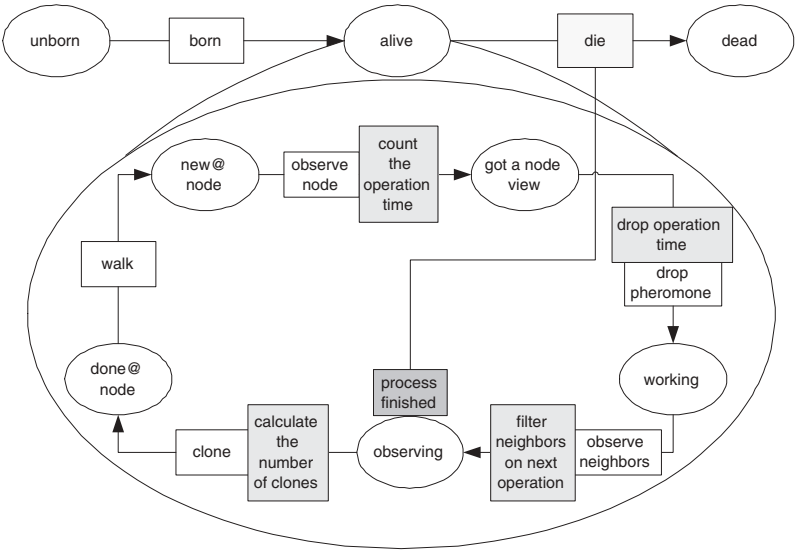

Fig. 2. An example of a statechart model.

– A final remark relates to the set-up of the statechart modeling language. Although different concerns of the agent's behavior can be modelled separately (in terms of building blocks provided by the statechart formalism), different concerns are mixed into one overall diagram. In the proposed statechart formalism no distinction is made between perceptions and actions in the environment, both are modeled as transitions. There is however a fundamental difference between these two activities. For a non-trivial agent merging the two in one model leads to poorly organized models. Another experience relates to the integration of coordination. In [20] we developed inter-agent coordination as a set of pre- and post-actions. The integration of the coordination in the agent's behavior model works well for rather simple agents, however for more complex cases, the models quickly become less surveyable. The underlying problem is that the integration of different concerns should be described separately of the concern descriptions themselves.

Other controlled techniques for engineering agent behavior have been applied such as Petri Nets, see e.g. [23][10][7], however the relationship between these techniques and our statechart modeling approach is out of the scope of this paper.

2.3 Combining the Best of Two Worlds

Agents must be able to adapt their behavior to deal with dynamism and change. Free-flow architectures enable adaptive behavior. However developing free-flow trees for non-trivial agents is a quasi-impossible task for software engineers. Architectures quickly becomes too complex to be manageable. To tackle complexity

suitable abstractions are needed to describe and structure the behavior of the agent. The role statechart modeling language offers a means to this. To combine the best of the two:

1. We extended the free-flow architecture with the abstractions of a role and a situated commitment.
2. We revised the statechart modeling language, i.e. we refrain from considering a statechart description of agent behavior as a kind of sequence chart, but use statecharts to describe role composition and to structure related actions in roles only.

As such statecharts structure the agent behavior reflected in the structure of the free-flow tree. Statecharts also provide an easy way to communicate description of the agent behavior at a higher level of abstraction.

3 Bringing the Statechart Models and Free-Flow Architectures Together

In this section we discuss the combined adoption of statecharts with the extended free-flow architecture. We illustrate our explanation with an example application. We start with a brief introduction of the example application. Next we describe the behavior of the agents with the statechart modeling language. Then we illustrate how the statechart models facilitate the structuring of a free-flow architecture.

3.1 Example Application

In a current research project with an industrial partner we investigate how the paradigm of situated MASs can be applied to the control of logistic machines. Traditional systems use one central controller that instructs the machines to perform jobs based on a calculated plan. The centralized approach lacks flexibility to deal with the increasing demands of adaptability and scalability. By looking at machines as agents of a situated MASs, we aim to convert the centralized control system into a self-managing distributed system, improving adaptability and scalability.

For the case in this paper we limit the discussion to the Automated Guided Vehicle (AGV) transport system. The AGV transport system is typically one, yet a crucial part, of an integral logistic warehouse system. AGV's are unmanned vehicles that transport goods from one place to another. AGV's can supply basic/raw materials to a production department, serve as a link between different production lines or store goods between different processes and connect to the dispatch area.

In a central controlled approach, the functionality of the individual AGV's is rather limited. Each AGV is provided with basic infrastructure to ensure safety, and a typical AGV is able to perform the pick and drop functionality

autonomously. The distribution of jobs, the routing through the warehouse, collision avoidance at junctions etc. are all handled by the central control system.

In this section we look at a number of basic roles for an AGV to deal with jobs autonomously. We take into account functionality for the AGV to find a job, to handle a job, to park when no more work has to be done and finally to ensure that the battery is charged in time.

3.2 AGV's Role Modeling

We distinguish two diagrams and one schema for role modeling. The *role diagram* structures the agent roles and their interdependencies. The *action diagrams* structure the related actions within the roles. Finally the *commitment schema* defines the activation and deactivation conditions for a situated commitment.

Role Diagram. The roles and their interdependencies that describe the behavior of an agent are described in a role diagram. Fig. 3 depicts the role diagram of the AGV's. A role diagram consists of a hierarchy of roles of which some are related through situated commitments.

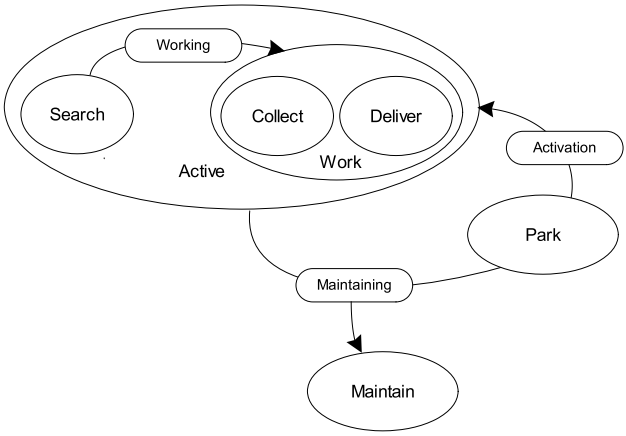

Fig. 3. The role diagram of the AGV.

A *role* is represented by a white oval and the name of the role is written in the oval. A role can consist of a number of sub-roles, and sub-roles of sub-sub-roles etc. As such the role diagram is typically composed as a hierarchy of roles. Roles at the bottom of the hierarchy are called *basic roles*. The first role of the AGV is the *Active* role consisting of two sub-roles, *Search*, i.e. a basic role, and *Work*. The *Work* role is further split up in two sub-roles, *Collect* and *Deliver*, two basic roles. In the *Search* role the AGV searches for a new job. Once the AGV finds a job it will *Collect* the good associated with the job and subsequently *Deliver* the good at the requested destination. Besides the *Active* role, the AGV has the *Maintain* role and the *Park* role. The AGV executes the

Park role when it has no more work to do. In this role the AGV simply moves to the nearest parking place. The *Maintain* role ensures that the AGV keeps its battery loaded. When the energy level crosses a critical value, the AGV finishes its current job and moves towards the nearest charging station. To find its way to the charging station an AGV uses an internal gradient map. At regular time intervals all charging stations broadcast their current status. AGV's use these messages to keep their gradient maps up to date.

A *situated commitment* is represented by a rounded rectangle and the name of the situated commitment is written in the rectangle. A situated commitment is defined as a relationship between one role, i.e. the goal role, and a non-empty set of other roles, i.e. the source roles of the agent. When a situated commitment is activated the behavior of the agent tends to prefer the goal role of the commitment over the source roles. Favoring the goal role results in more consistent behavior of the agent towards the commitment. An agent can commit to itself, e.g. when it has to fulfill a vital task. However, in a collaboration, agents commit relatively to one another, typically through communication. [34] discusses mutual commitments between collaborating agents in detail. In Fig. 3, the *Maintaining* commitment ensures that the AGV maintains its energy level. Since energy is vital for the AGV to function, all roles (except the *Maintain* role of course) are connected as source roles to the *Maintaining* commitment. The *Activation* commitment is activated when the AGV starts to work. This commitment ensures that the AGV remains active once it decides to start working. *Working* is an example of a commitment in a collaboration. The commitment is activated once the AGV accepts a job. This commitment ensures that the AGV acts consistently with the job in progress. As soon as the job is finished, the *Working* commitment is deactivated.

Action Diagram. Action diagrams are defined for the basic roles. An action diagram describes the structure of the related actions for a basic role. In Fig. 4 the action diagram of the *Maintain* role of the AGV is depicted.

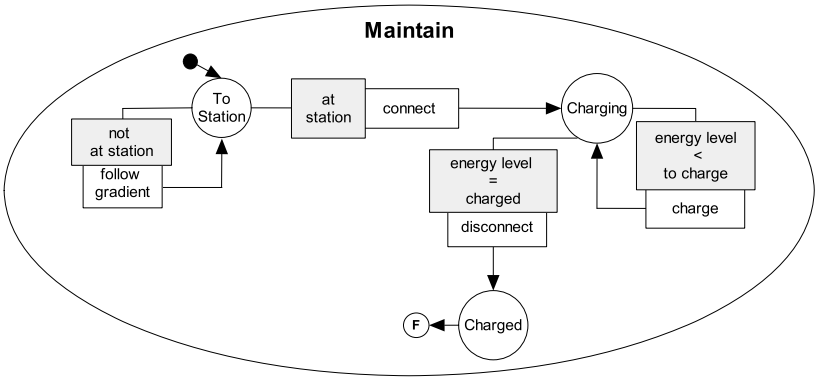

Fig. 4. Action diagram of the Maintain role.

A *state* is represented by a white circle in the diagram. In Fig. 4 three states can be distinguished: *ToStation*, *Charging* and *Charged*. Besides regular states there are two special states. The *default state*, represented by a black circle, indicates the typical start state of the action sequence of the modelled role. On the other hand, there is the *final state*, represented by a circle with an F written in it, that indicates the typical end state of the action sequence of the modelled role. The default and final state are connected to the corresponding regular state via an arrow.

A *transition* connects two states with each other. A transition expresses a change of state due to the execution of an action. An *action*, which is added to a transition, models the functionality that must be performed by an agent to achieve a new desired state from an old state. An action is represented by a white rectangle in which the name of the action is written and which is attached to a transition. To fulfill the *Maintain* role the AGV has to perform four different actions: *follow gradient* to find the charge station, and *connect*, *charge* and *disconnect* to charge the battery (see Fig. 4). The execution of an action may be constrained by a *precondition*. Only when the condition is satisfied the attached action can be executed. A precondition is represented by a gray rectangle in which the precondition is written and which is attached to an action. In Fig. 4 the gray rectangle with *not at station* denotes that the AGV keeps following the gradient until it reaches the charge station. At that time the precondition *at station* becomes true and that enables the AGV to *connect* to the charge station. As long as the condition *energy level* $<$ *charged level* holds, the AGV keeps charging. Finally when condition *energy level* $=$ *charged level* becomes true, the AGV *disconnects* and that finishes the *Maintain* role.

Commitment Schema. For each situated commitment a commitment schema is defined that describes the source roles and the goal role of the commitment as well as its activation and deactivation conditions. Activation and deactivation conditions are boolean expressions based on the internal state of the agent, perceived information or information derived from received messages. Activating situated commitments through communication enable situated agents to setup explicit collaborations in which each participant plays a specific role. In this paper we do not elaborate on this latter scenario, for a detailed discussion we refer to [34]. Fig. 5 depicts the commitment schema for the situated commitment *Maintaining*. This commitment schema expresses that when the *energy level* of the AGV falls below the threshold *to charge* the situated commitment $Maintaining$ is activated. This will urge the AGV to execute the $Maintain$ role over the *Active* and $Park$ roles. Once the battery is recharged the condition *energy level* $=$ *charged* becomes true and that deactivates the $Maintaining$ commitment.

3.3 Free-Flow Architecture

The free-flow tree describes the behavior of the agent in detail. The high-level diagrams for roles and situated commitments described in the previous section

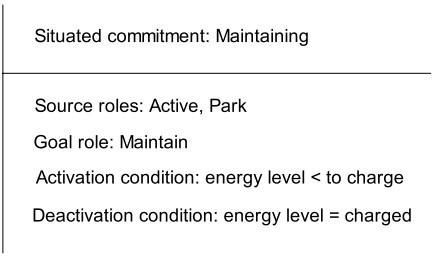

Fig. 5. The commitment schema for the situated commitment Maintaining.

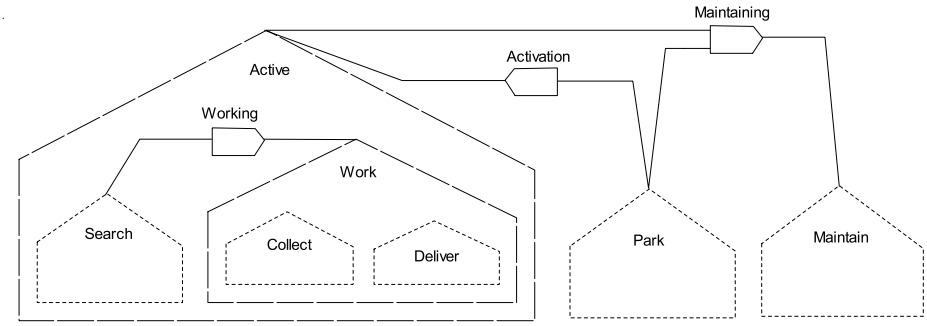

Fig. 6. Skeleton structure of the free-flow tree according to the role diagram of Fig. 3.

serve as a basis for structuring the free-flow tree. The role structure as described in the role diagram (see Fig. 3) is reflected in the skeleton structure of the tree. Fig. 6 depicts the skeleton structure for the AGV example. Roles match to trees in the free-flow tree, sub-roles to sub-trees etc. Situated commitments on the other hand corresponds to connectors that connect the source roles of the situated commitment with the goal role. When a situated commitment is activated extra activity is injected in the goal role relative to the activity levels of the source roles. Details are discussed shortly.

The action diagrams and commitment schemas enable to refine the skeleton tree. Fig. 7 depicts the refined sub-tree for the *Maintain* role and the *Maintaining* commitment.

States in the action diagram correspond to activity nodes in the tree. Preconditions correspond to binary stimuli connected to the corresponding nodes. Examples are the stimuli *at station* or *connected* (compare Fig. 4 and Fig. 7). Each action in the action diagram of the basic role corresponds with an action node in the tree. A number of other non-binary stimuli in the tree represent data in the action diagram that determines the action selection. An example is the stimulus *gradient* that guides the AGV to move towards the station.

The activation and deactivation conditions of the situated commitments, described in the commitment schema, correspond to the conditions associated

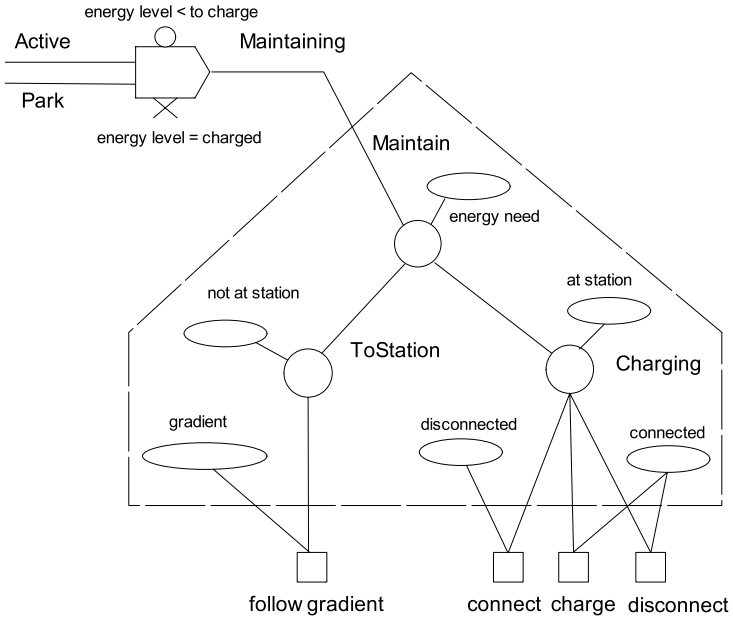

Fig. 7. Refined Maintain role and Maintaining commitment.

with the corresponding connectors in the free-flow activity tree. Fig. 7 illustrates this for the *Maintaining* commitment.

3.4 The Complete Free-Flow Tree

The complete free-flow contains all detailed information needed for action selection. Fig. 8 depicts the completed subtree of the *Maintain* role and the situated commitment *Maintaining*. The abstract action node *follow gradient* in Fig. 7 is refined towards the different moving actions of the AGV. The stimulus *gradient* is split up in a multi-directional stimulus. Each segment represents the tendency (based on the value of the gradient field) of the AGV to move in a particular direction. Besides, a number of extra stimuli represent data that influences the action selection. An example is the multi-directional stimulus *free* that denotes in which direction the AGV is able to drive.

Stimuli needed to verify the activation and deactivation condition are connected to the situated commitment. The *Maintaining* commitment is activated when the value of the *energy level* crosses the threshold *to charge*. The commitment then calculates the extra activity to inject in the *Maintain* role. For the *Maintaining* commitment this extra activity is calculated as the sum ("+" symbol) of the activity level of the *Active* and *Park* role, i.e. the activity levels of the top nodes of these roles. As soon as the battery level reaches the threshold value *charged* the *Maintaining* commitment is deactivated and it then no longer injects extra activity in the *Maintain* role.

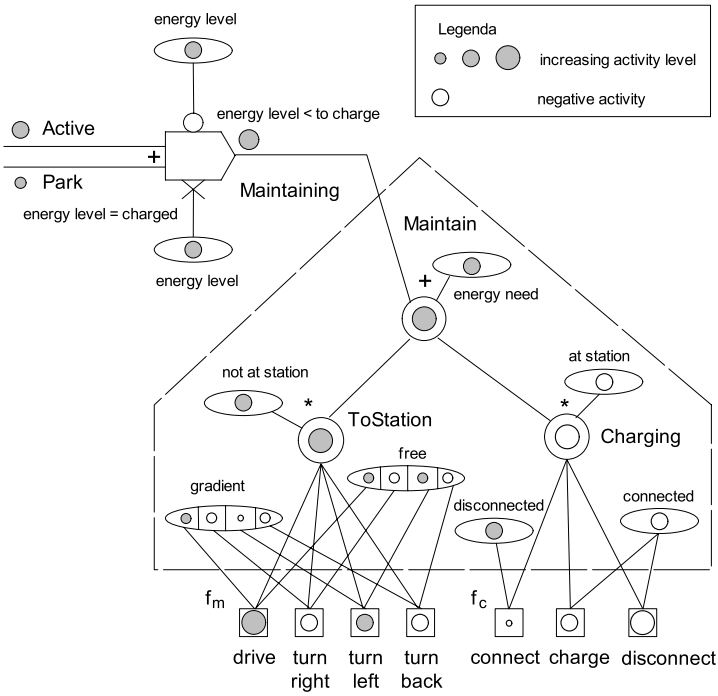

Fig. 8. The complete Maintain role and Maintaining commitment.

4 Discussion

This paper introduces a practical approach to combine adaptiveness of agents and MAS with rigid/controlled engineering. The approach enables engineers to manage the complexity of designing free-flow architectures. The proposed role abstraction allows to represent local agent activity. Roles however, not only "chop up" the behavior of the agent into smaller logical parts, they also introduce a means to enable explicit collaboration between situated agents, reified in situated commitments. In the AGV case e.g., when an *Searching* AGV accepts a job, it activates the *Working* commitment and that will bias the action selection of the AGV towards the *Work* role. As such, the AGV will act consistently towards its commitment in the collaboration, i.e. its agreement to perform the job in progress.

The focus of this paper is mainly on the *integration* of free-flow architectures with role modeling based on statecharts. In ongoing work [30] we described a design process for adaptive agent behavior as part of a multi-agent oriented methodology. This process integrates the engineering approach for behavior design we have proposed in this paper and rigorously describes the subsequent design steps. At the highest level, roles and their interdependencies are caught into a high level model described making use of the statechart modeling language. This model is used as a basis for designing a skeleton of the free-flow

architecture. Next the skeleton is refined such that it contains all details needed for action selection. Finally, the free-flow tree is mapped onto a class diagram that serves as a basis for the implementation of the agent's behavior.

Several agent-oriented methodologies acknowledge the abstraction of a role as a core abstraction for designing multi-agent systems, examples are Gaia [36], MESSAGE [8] or SODA [26]. In these methodologies the design process is described independent of a particular multi-agent architecture, for a recent discussion see Chapter 4 of [21]. When it comes to building a concrete multi-agent application however, the gap between the high level design models and the chosen multi-agent architecture that is used to implement the multi-agent system has to be filled. We aim to bridge this gap enabling designers to build concrete multi-agent systems applications. In particular, the design process described in [30] that builds upon the software engineering approach for behavior design proposed in this paper, enables a designer to incrementally refine the model of the agent behavior from a high level role model toward a concrete agent architecture for adaptive behavior, in casu a free-flow architecture.

5 Conclusion

Engineering software for non-trivial open multi-agent systems is a challenging task. In this paper we proposed a software engineering approach that combines free-flow architectures for adaptive behavior with a statechart modeling language that offers suitable abstractions. Free-flow trees are extended with the abstraction of a role and a situated commitment. The earlier developed statechart formalism is revised and adapted from a rigid description of action sequences towards a description of the role composition of the agent behavior and a structuring of the related actions within the roles. In the paper we illustrated the approach for a case study on controlling a collection of automated guided vehicles.

Currently we are working on a design process for adaptive agent behavior that integrates the engineering approach for behavior design we have proposed in this paper. In future work we intend to extend the design process towards other concerns that need to be engineered in situated MASs such as agent communication and coordination, and the design of the environment of the MAS.

Acknowledgement

The research results presented in this paper have been obtained in the Concerted Research Action on Agents for Coordination and Control - AgCo2 project (K.U.Leuven) and in the Egemin Modular Controls Concept - EMC2 project (Flemish Institute for the Advancement of Scientific-technological Research in the Industry - IWT).

References

1. Babaoglu, O., Meling, H., Montresoret, H.: Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems. International Conference on Distributed Computing Systems, Vienna, Austria (2002)
2. Balch, T., Arkin, R.C.: Communication in Reactive Multiagent Robotic Systems. Autonomous Robots **1(1)** (1995)
3. Bonabeau, E., Henaux, F., Guerin, S., Snyers, D., Kuntz, P., Theraulaz, G.: Routing in Telecommunications Networks with Ant-Like Agents. IATA (1998)
4. Brooks, R.: Intelligence without representation. Artificial Intelligence Journal, Vol. 47 (1991)
5. Brooks, R.: Intelligence Without Reason, MIT AI Lab Memo No. 1293 (1991)
6. Bryson, J.: Intelligence by Design, Principles of Modularity and Coordination for Engineering Complex Adaptive Agents. PhD Dissertation, MIT (2001)
7. Cabac, L., Moldt, D.: Formal Semantics for AUML Agent Interaction Protocol Diagrams. 5th International Workshop on Agent-Oriented Software Engineering, AOSE at AAMAS, New York (2004)
8. Caire, G., Leal, F., Chainho, P., et al.: Agent Oriented Analysis Using MESSAGE/UML. Agent-Oriented Software-Engineering II, Lecture Notes in Computer Science, Vol. 2222, Berlin Heidelberg New York, Springer (2001)
9. Cernuzzi, L., Juanand, T., Sterling, L., Zambonelli, F.: The Gaia Methodology: Basic Concepts and Extensions. Methodologies and Software Engineering for Agent Systems, Kluwer (2004)
10. Cost, R.S., Chen, Y., Finin, T., Labrou, Y., Peng, Y.: Modeling agent conversations with colored petri nets. Workshop on Specifying and Implementing Conversation Policies, Seattle, Washington (1999)
11. Deneubourg, J.L., Aron, A., Goss, S., Pasteels, J.M., Duerinck, G.: Random Behavior, Amplification Processes and Number of Participants: How they Contribute to the Foraging Properties of Ants. Physics **22(D)** (1986)
12. Dorigo, M., Gambardella, L.M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation **1(1)** (1997)
13. Drogoul, A., Ferber, J.: Multi-Agent Simulation as a Tool for Modeling Societies: Application to Social Differentiation in Ant Colonies. Decentralized A.I. **4**, Elsevier North-Holland (1992)
14. Ferber, J.: An Introduction to Distributed Artificial Intelligence. Addison-Wesley (1999)
15. Ferber, J., Gutknecht, O., Michel, F.: From Agents to Organizations: An Organizational View on Multi-Agent Systems. 3th International Workshop on Agent Oriented Software Engineering, AOSE, Melbourne, Australia (2003)
16. Genesereth, M.R., Nilsson, N.: Logical Foundations of Artificial Intelligence, Morgan Kaufmanns (1997)
17. Griss, M.L., Fonseca, S., Cowan, D., Kessler, R.: Using UML State Machine Models for More Precise and Flexible JADE Agent Behaviors. 2th International Workshop on Agent Oriented Software Engineering, AOSE, Bologna, Italy (2002)
18. Harel, D.: Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming **8(3)** (1987)
19. Holvoet, T., Steegmans, E.: Application-Specific Reuse of Agent Roles. Software Engineering for Large-Scale Multi-Agent Systems, Lecture Notes in Computer Science, Vol. 2603, Berlin Heidelberg New York, Springer (2003)

20. Janssens, N., Steegmans, E., Holvoet, T., Verbaeten, P.: An Agent Design Method Promoting Separation Between Computation and Coordination. Symposium on Applied Computing SAC, Nicosia, Cyprus (2004)
21. Luck, M., Ashri, R., D'Inverno, M.: Agent-Based Software Development. Artech House (2004)
22. Maes, P.: Modeling Adaptive Autonomous Agents. Artificial Life Journal **1(1-2)** (1994)
23. Ferber, J., Magnin, L.: Conception de systemes multi-agents par composants modulaires et reseaux de Petri. Actes des journees du PRC-IA, Montpellier (1994)
24. Odell, J., Parunak, H.V.D., Bauer, B.: Extending UML for Agents. AOIS Workshop at AAAI, www.auml.org (2000)
25. Odell, J., Parunak, H.V.D., Fleisher, M.: The Role of Roles. Journal of Object Technology **2(1)** (2003) http://www.jot.fm/
26. Omicini, A.: SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems. Agent-Oriented Software Engineering, Lecture Notes in Computer Science, Vol. 1957, Berlin Heodelberg New York, Springer (2001)
27. Parunak, H.V.D.: Go to the Ant: Engineering Principles from Natural Agent Systems. Annals of Operations Research **75** (1997)
28. Parunak, H.V.D.: The AARIA Agent Architecture: From Manufacturing Requirements to Agent-Based System Design. Integrated Computer-Aided Engineering **8(1)** (2001)
29. Rosenblatt, K., Payton, D.: A fine grained alternative to the subsumbtion architecture for mobile robot control. International Joint Conference on Neural Networks, IEEE (1989)
30. Steegmans, E., Weyns, D., Holvoet, T., Berbers, Y.: Designing Roles for Situated Agents. 5th International Workshop on Agent-Oriented Software Engineering, AOSE at AAMAS, New York (2004)
31. Steels, L.: Cooperation between distributed agents through self-organization. Proceedings of the First European Workshop on Modeling Autonomous Agents in a Multi-Agent World, Elsevier Science Publishers, Holland (1990)
32. Tyrrell, T.: Computational Mechanisms for Action Selection. Ph.D Thesis, University of Edinburgh (1993)
33. Weyns, D., Holvoet, T.: A Formal Model for Situated Multi-agent Systems. Formal Approaches for Multi-Agent Systems, Special Issue of Fundamenta Informaticae, **63(2–3)** (2004)
34. Weyns, D., Steegmans, E., Holvoet, T.: Protocol Based Communication for Situated Multi-agent Systems. 3th International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS, New York (2004)
35. Wooldridge, M., Jennings, N., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Autonomous Agents and Multi-Agent Systems **3(3)** (2000)
36. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing Multiagent Systems: The Gaia Methodology. ACM Transactions on Software Engineering and Methodology **12(3)** (2003)