A Methodology for the Protocol-Centered Design of Agent-Based Systems

P.C.P. Bhatt
Information Systems Engineering
Kochi University of Technology
Kochi 782-8502, Japan
bhatt@info.kochi-tech.ac.jp

W. Mueller
C-LAB
Paderborn University
D-33102 Paderborn, Germany
wolfgang@acm.org

Abstract

This article presents a novel methodology for the protocol-centered design of agent-based systems. Agents are understood as concurrently interacting autonomous, proactive objects in a distributed environment. The introduced methodology is diagram based and focuses on the initial phase of specifying and exploring communication protocols between agents. It is based on the derivation of protocols from actors and their events. We additionally provide design rules and conventions for the seamless transition of the specification to an implementation. The concepts are presented through an example from Computer Integrated Manufacturing (CIM) which gives the specification of autonomous, automated guided vehicles (AGVs) moving parts between manufacturing workstations.

1 Introduction

With increasing automation, manufacturing systems are currently moving towards distributed computer-based systems. In this context agent-based systems technologies for the implementation of distributed systems are of emerging interest. The agent-based paradigm can be seen as a real enhancement of the object-oriented paradigm where objects become autonomous, communicative, and proactive. The general correctness of a distributed agent-based system largely depends on the correct specification, implementation, and testing of the involved protocols. A careful investigation of protocols in initial phases of the design process helps in plugging them together with relative ease and fewer problems during later design phases.

Though computer-based systems have evolved to distributed systems, we surprisingly find only very few methodologies for early design phases which are dedicated to protocol design. UML only partly supports protocol design through sequence and state diagrams [6]. The given UML development process seems to be too general to provide real support for such an application. Wooldridge et al. recently have proposed a methodology for the design of agent-based systems which covers protocol design but only gives little support for the composition of agents; refinement of agents’ behaviour is not sufficiently worked out [9]. However, due to its flexibility and coverage we found that the Coats-Mellon Operational Specification (CMOS) [4] provides a couple of adequate principles and means for our work.

We present a novel methodology for the protocol-centered design of distributed, agent-based systems. The methodology addresses the first phases of the design process mainly focusing on manual analysis. It gives input to latter phases where automated or semi-automated analysis like simulation or model checking can be applied. The approach is partly based on the CMOS development process and its diagrams with actor, event category, and actor event diagrams. Additionally, we give detailed design conventions and rules to finally arrive at a state-oriented protocol implementation. The work is presented through a realistic problem from holonic manufacturing systems (HMS) [8]. Our investigations have shown that the presented methodology provides adequate means to faster arrive at a first implementation (or executable specification) and to easier detect protocol errors.

The remainder of this paper is organised as follows. We first discuss related methodologies. Thereafter, we introduce the basic principles of an automated guided vehicle in a holonic manufacturing system. Section 4 presents our methodology by the means of the example
of a battery charge request for such a vehicle. The paper closes with a short summary and conclusion.

2 Related Design Methodologies

Since a majority of todays system development activities are moving towards UML (Unified Modelling Language) we will take a brief look at UML first. Presently, only very few approaches are dedicated to agent-based systems. The most practical one seems to be the one from Wooldridge et al. which is discussed thereafter. We finally investigate the Coats-Mellon Operation Specification (CMOS) from which our methodology largely inherits from.

2.1 UML

UML has a very rich notational framework which is deeply embedded in, and tied to, object oriented methodology. UML is most useful for communication amongst designers, and design teams, to understand and explain designer’s intent.

UML expresses models through a rich set of diagrams, i.e., class, package, deployment, use case, collaboration, sequence, activity, and state diagrams.

UML comes with a development process which is basically given as a matrix with phases (inception, elaboration, construction, transition) and workflow activities (requirements, analysis, design, implementation, test). Due to its intended genericity it does not give very precise guidelines on how to organise an application-specific UML-based development process.

Recent developments in Real-Time-UML, Automotive UML, and Agent UML demonstrate that Standard UML has deficiencies for the development of distributed systems. As shown in the next section, sequence diagrams actually need to be enhanced to cover additional needs. State diagrams, mainly due to their synchronous semantics with broadcast, are not adequate for the specification of the interaction of distributed system components.

2.2 Agent-Based Design

Wooldridge et al. have introduced an agent-oriented analysis and design methodology in [9]. The analysis process has three steps.

- identify agents with roles and keyroles
- for each role identify a protocol by an interaction model with purpose, initiator, inputs, outputs
- elaborate roles with permission, responsibilities including behaviour

The design process has three steps.

- aggregate roles and refine to form an agent type hierarchy
- elaborate roles focusing on behaviour and constraints
- define communication links between agent types

The methodology comes with few diagrams like the roles diagram and forms specifying agent behaviour, i.e., the agent’s protocol. One important point in this methodology is the consideration of safety (constraints) and access permission. However, we see that the given protocol notation is not suitable for the specification of larger protocols that require exception handling and error recovery. Furthermore, composition of protocols is not supported. We feel that for general system development the identification of communication links has to be considered in the earlier phases. Since component-oriented development has to capture existing components it requires an early investigation of their integration.

2.3 CMOS

An operational approach to systems design is given with CMOS (Coats-Mellon Operational Specification). In CMOS, the first step is the identification of actors within a system. Basically, the system is modelled around the responses to actor-generated stimuli. Actors usually are identified with the devices and entities that interact. To that extent operational specifications using CMOS stipulate a closed world with known actors.

Like the UML notation, the CMOS approach too requires a set of diagrams to be drawn with a specific set of notations as given in Figure 1. The CMOS development process is organised as follows.

CREATE AN ACTOR DIAGRAM: This requires that entities that play a role are identified; arrows depict the
direction of communication. In the example of Figure 1, the two actors have a connecting bidirectional arrow.

**CREATE AN ACTOR INHERITANCE DIAGRAM:** This requires that we identify objects which inherit properties from generic actor class definitions.

**CREATE AN EVENT CATEGORY DIAGRAM:** This idea is very similar to the identification of activities in UML. Though it should be remarked that the event category lists the events and event sequences as a catalogue (see Fig. 1).

**CREATE AN ACTOR EVENT DIAGRAM:** Actor event diagrams are like unfolded UML sequence diagrams but often capture a feed back explicitly. In Figure 1 we have shown a typical actor event diagram for an AGV docking with an input station where we first check the id of the AGV and proceed to dock only when the id is ok, else following a sequence identified with an error recovery routine.

**CREATE SYSTEM RESPONSE DIAGRAMS:** System response diagrams are high level descriptions and are more like a wish list for the final system. After having created all diagrams CMOS recommends to validate the behaviour.

We basically found that these principles are in general most adequate for the design of distributed agent-based systems since protocol-oriented concepts with refinements are supported.

# 3 Holonic Manufacturing

Our running example specifies a holonic manufacturing system (HMS) which was introduced by the IMS* Initiative TC 5 [8]. An HMS is based on the notion of a holon which denotes a building block of a system for complex production lines. Hierarchically composed holons cooperate along the lines of basic rules to achieve a goal or objective. A single holon is understood as an autonomous cooperative building block which mostly consists of an information processing and a physical part. The holonic system structure is based on the principles of self-similarity and self-configuration which form self-adaptable and thus fault-tolerant networks of holons.

Our HMS example is composed of a set of different manufacturing work stations and a transport system as it is given in the virtual 3D model in Figure 2. The different manufacturing stations transform parts by, for instance, milling, drilling, and washing. Additional input and output stations are for primary system input and output. The transport system consists of a set of autonomous AGVs (Automated Guided Vehicles) which move parts between manufacturing work stations (MWS). An AGV

1. is idle until it receives a request for delivery from a station $s_i$
2. sends the costs $c_i$ to get from its current position to $s_i$
3. moves to $s_i$ on notification of acceptance from $s_i$
4. takes the part from the output and moves it to its destination
5. moves to parking position and continues with 1.

AGVs are battery-driven. If detecting low battery level, an AGV calls for an idle battery (charging) station. After their replies, the AGV selects one and moves to that station for charging.

Stations have input and output units for incoming and outgoing parts. Once having one part at the output unit, a station

1. broadcasts a request for delivery to all AGVs
2. receives costs $c_i$ from idle AGVs for a specific time period
3. continues with Step 1 if no AGV replies
4. selects one AGV computing all received costs values $c_i$ and their reliability factor $r_i$
5. notifies that AGV for acceptance and notifies other AGVs for rejection

# 4 Protocol-Centered Methodology for Agent-Based Systems

We present a novel methodology for the protocol-centered design of distributed, agent-based systems. The methodology starts from scratch with the identification of agents (i.e., actors) and their communication structure. We arrive at the final specification

---

*IMS (Intelligent Manufacturing Systems) is an industry-led international research & development program to develop the next generation of manufacturing and processing technologies with currently over 250 companies and over 200 research institutions participating.
of state-oriented communication protocols through a stepwise approach. In details, the individual design steps are

1. identify actors and their basic communication
2. identify events for each actor
3. identify communication between actors
4. specify actors with communication & local behaviour with clear identification of the initiator for each protocol
5. refinement of steps 1. - 4.

As an example, the remainder of this section outlines a protocol which negotiates battery charge requests of an AGV with battery (charging) stations.

### 4.1 Actor Identification

As a first step, we have to identify actors and their communication. For this we basically apply CMOS actor diagrams with additional hierarchical decomposition of actors into their subunits. Thus, we first identify the main actor classes as well as their subunits as far as they are relevant for the level of abstraction. Thereafter, we identify their basic interactions by sketching them by uni- or bidirectional lines between the actor classes. Individual applications may consider access rights or permissions for each communication, if required.

Figure 3 gives the actor diagram of the previously introduced holonic manufacturing system. We can identify the input station (IN), output station (OUT), vehicle (AGV), manufacturing workstation (MWS), and battery charging station (BS). The AGV has as subunits a loading unit (LU), an unloading unit (UU), and a processing unit (PU). The MWS has an input unit (IU), output unit (OU), and a processing unit (PU). Processing units are in both cases understood as information processing units which are also in charge of the communication between actors. The MWS-PU additionally stands for the combined abstraction with its physical manufacturing unit.

![Figure 3: Actor Diagram](image-url)

### 4.2 Event Identification

For each of the actors we identify their tasks at event level. For each actor we draw one event category diagram. A category diagram has the actor at its root and defines the tasks in different branches. At the leaves we have an enumeration of the basic events within a task. Only those events are enumerated which are relevant for communication. This diagram provides initial means to start the exploration of the communication of actors from. Event category diagrams can be roughly seen as enhanced UML use case diagrams with the additional enumeration of events.

Figure 4 gives the event category diagram of an AGV-PU (Processing Unit) for Charge Battery. We identify two main events: (i) Charge Request and (ii) Charge. The first task (Charge Negotiate) sends a request for charging to all battery stations. After waiting for their replies the AGV sends an Accept to one station and a Reject to the others. Note that we only consider the system from the perspective of the AGV-PU here. The events of the communication partner is defined in the corresponding BS event category diagram.

![Figure 4: Event Category Diagram](image-url)

### 4.3 Actor-Event Communication

After having explored all events, we define the individual protocols as a set of actor event diagrams for each actor, one for each protocol. Actor event diagrams only consider the actors and events which have been identified in the actor diagram and the event category diagrams before. Firstly, we assign the roles of initiators and responders to actors for each protocol. A protocol definition starts with the initiator as root. It is drawn as an actor event diagram from left to right with actors as nodes and events as lines. In order to increase the readability of the diagram, actors are placed on vertical bars which are numbered from left to right. Feedbacks are denoted by circles which give a return address to a bar. Note here, that actor event diagrams can be hierarchically organised, i.e., a node can enclose a diagram again.
Figure 5: Actor Event Diagram

Figure 5 shows the definition of the ChargeRequest protocol between the AGV-PU and the BS. The AGV-PU as the initiator of the protocol sends a ChargeRequest to all battery stations. The battery stations either send a Free message or a Blocked message when they are already reserved for another AGV. The AGV-PU selects one station and sends an Accept message in return. All others receive a Reject message.

With increasing complexity of protocols CMOS actor event diagrams are clearly ahead of UML sequence diagrams since latter can capture only one trace in one diagram. For more complex communication structures the user has to draw a set of separate sequence diagrams instead of having all information combined in one. Figure 6 gives an example. It has all sequences of the Figure 5 CMOS diagram combined in one sequence diagram. Note that this diagram is actually invalid UML since we would need to draw three diagrams, one for each sequence: ChargeRequest|Free|Accept, ChargeRequest|Free|Reject, ChargeRequest|Blocked.

Figure 6: Combined UML Sequence Diagram

4.4 Actor Specification

While the previous step has explored the different protocols in some details we now focus on the refinement of the actor’s behaviour. Following the principle of separation of concerns, we distinguish three different aspects of an actor: user interface, local behaviour, and communication. Figure 7 gives a basic template for the definition of actors. User interface is optional and can be given by simple devices like switches, LEDs, or more complex devices like monitors or keyboards. At this stage, it is only specified that something is displayed or expected as input but not how. Note here, that we have chosen to model sensors and actors as separate agents which interact via the communication interface with the local behaviour, i.e., controller. Alternative designs may include them in the local view.

Figure 7: Separation of User Interface, Behaviour, Communication

Let us focus on the refinement of the local behaviour of an actor now. We start from the event category diagram in which the event sequence is given. Each event in the event sequence is mapped to one major state which are drawn as single lines labelled by the event name, so that the event name becomes the identifier of the major state. Thereafter, we take the corresponding protocol given as event actor diagram and draw all possible state transitions as arrows between the lines. The so derived definition specifies a first local state machine which drives the required protocol for the individual actor. Figure 8 shows the state machine for a simple version of AGV’s ChargeNegotiate protocol.

Figure 8: State Diagram for ChargeNegotiate

In the next step, the state machine is enhanced with communication and user interface interaction by the use of the previously introduced template so that we arrive at an initial specification of that protocol as shown in Figure 9. Arrows indicate messages or events which are received or sent via the communication interface. On the left, major states are associated with

---

2Alternatively, it is also possible to use any comparable state-oriented means like ITC SDL Process Diagrams.
the user interface devices. In our example, LED activity shall indicate activity when exchanging messages.

![LED Diagram]

**Figure 9: AGV Specification**

Therefore, the (local) state machine has to be refined by introducing intermediate states for local processing like the computation of values etc. Therefore, we finally have to

1. combine the different protocols of one actor with its local behaviour
2. define minor states within major states
3. assign the communication policy

The order in which those steps are applied heavily depends on the chosen implementation language and on the individual application. Parallel languages certainly ease the implementation here. In the first step, different protocols can be mapped to different parallel state machines given by different UML state diagrams, for instance. Of potential application are also the means of extended Petri-Nets like Morphic Nets [2], Abstract State Machines [7], or Esterel [1]. In a second step, we refine the specification of the local behaviour to an implementation. The third step considers the composition of actors to a system as given by the actor diagram. Communication policies like point-to-point, multi-cast, broadcast have to be assigned to each communication channel.

Actor diagrams in combination with the specification of local behaviour provide a complete specification of the distributed system from which a first implementation can be easily derived and which are candidate to further refinement.

## 5 Summary and Conclusion

We have presented an approach how to enhance the CMOS methodology and its means for protocol-centered design. We have achieved quite reasonable results in the specification, design, and implementation of the HMS which was described in Section 3. We have applied the presented methodology to redesign a Tcl/Tk prototype implementing the protocol for delivery negotiation. The virtual 3D model in Figure 1 gave us a test platform for that implementation [5]. During that work we could detect a design error that one protocol was not correctly initiated, i.e., the role of the initiator was not clearly identified. That experiment also demonstrated that our methodology provides a seamless transition from specification to implementation.

## Acknowledgments

We are grateful to Prof. Franz J. Rammig, Stephan Flake, and Georg Lehrenfeld from Paderborn University for their continued support. The presented work is partly funded by the GRASP project under the DFG programme 1064 "Integration of Software Specification Techniques for Engineering Applications".

## References


