

# AFDM: A UML-based Methodology for Engineering Intelligent Agents

Yanjun Tong, G. M. P. O'Hare, Rem Collier

Department of Computer Science, University College Dublin, Ireland

**Abstract.** Agents are a potential technology with many applications. It is urgent to develop appropriate methodologies for the development and deployment of agent-oriented applications. This paper presents Agent Factory Development Methodology (AFDM), a cohesive methodology, to support the development of agent-oriented applications. On the other hand agent cloning is considered one of the possible approaches to avoid system overloading. In later sections of this paper, agent cloning is designed and implemented as an example through the AFDM.

## 1 Introduction

With the continuing emergence of the Agent-Oriented paradigm, it is urgent to develop appropriate methodologies for the development and deployment of agent-oriented applications. This paper motivates and presents the Agent Factory Development Methodology (AFDM) for the design and development of intelligent agent systems. This methodology has been employed in the development of a number of large scale agent-based applications including the WAY System[13], Gulliver's Genie[7] and the award-winning ACCESS Architecture[3].

The Agent Factory System is a cohesive framework that delivers structured support for the development and deployment of agent-oriented applications. It is comprised of four distinct layers, namely: Agent Factory Programming Language (AF-APL), Agent Factory Run Time Environment (RTE), the Agent Factory Development Environment and the Agent Factory Development Methodology. The RTE and AF-APL, which provide supports for agent migration, agent management and resource management and reside at the core of the system. Figure1 presents a demonstration of the AF framework.

The AF-APL represents a dedicated Agent Oriented Programming language with logical, deliberative and imperative features. Agent within Agent Factory [4] can be considered as an amalgam of three sets of components: the mental state, actuators, and perceptors.

The mental state contains the agent's current model of itself and the environment it is situated. This knowledge is stored as 'Beliefs'. Each 'Belief' represents a piece of knowledge an agent thinks it knows about the environment. For instance Belief(like(rice)) can be interpreted as meaning 'the agent likes rice'.

Perceptors are those components that convert raw sensory data into beliefs which are subsequently adopted and augmented into the existing into the agents'



**Fig. 1.** AF Framework

belief set. For example an agent may have a perceptor which obtains the current time and generates a belief like: `Belief(curreTime(17:00 pm))`.

Within AF-APL, actions are realized through the triggering of an associated actuator unit. As with perceptor units, actuator units are associated with specific agents as part of the agent program through the `ACTUATOR` keyword. Actions can be combined into plans that form more complex behaviors by using one or more plan operators - currently there are four plan operators: sequence (`SEQ`), parallel (`PAR`), unordered choice (`OR`), and ordered choice (`XOR`). For example an actuator which clones an agent may be triggered by the action 'Clone'. Details about `BELIEF`, `ACTUATOR` and `PERCEPTOR` are presented in [10].

One of the primary objectives behind AF is the development of a cohesive software engineering methodology that delivers structured support for the design, implementation, and deployment of multi-agent systems. In designing this methodology, we sought to address a number of objectives: (1) to employ, where possible, pre-existing industry recognisise design notations; (2) to focus upon the definition of visual notations; (3) to use models that promote design reuse; and (4) to maintain a strong link between design and implementation, opening the way for automated code generation.

Within Multi-Agent Systems (MAS), it is imperative that task distribution strategies take cognizance of load-balancing issues. Load-balancing mechanisms thus need to be formulated and delivered which prevent individual agents and/or processors becoming over-loaded. Agent migration is one key technology which may be commissioned in this regard. Another mechanism is that of imbuing individual agents with the autonomy to invoke and negotiate load-balancing regimes. Agent cloning is considered as one instrument and possible responses to the above approach. Embedded with cloning ability, when over-loaded, an agent may generate several concurrent agent instances with the same mental

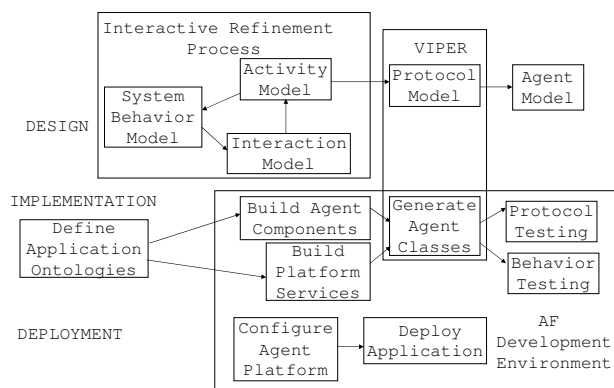
states and behaviors, and discharge the computing tasks to appropriate clone instances.

The work presented in this paper is concerned with the design and implementation of an agent cloning capability. In particular, we adopt the Agent Factory Development Methodology in the design of the agent cloning model (section 2). Section 3 highlights the use of this methodology in agent cloning as an exemplar case study. Finally, Section 4 introduces some related work and gives some concluding marks.

## 2 The Design Phase of Agent Cloning

The design phase is concerned with the translation of system requirements into a well defined model of the target system that may easily be implemented using agent technologies. Five key models have been developed within the Agent Factory Development Methodology layer. These models include the System Behaviour Model, the Interaction Model, the Activity Model, the Protocol Model and the Agent Model [5]. Figure 2 presents a diagrammatic overview of this methodology.

However, since both the Interaction Model and the Protocol Model concentrate on the interplays between agents, there is no need to design both of them. Consequently in the later sections we consider each of the system behavior, activity, protocol and agent models in turn.

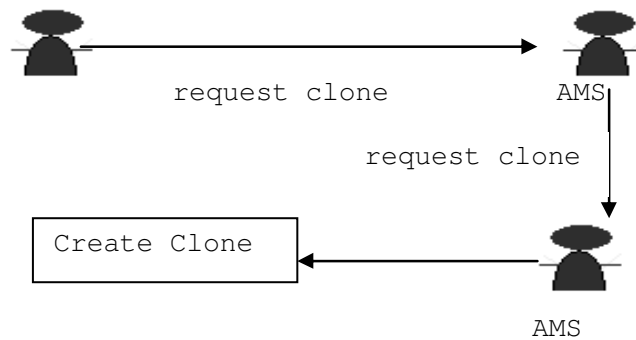


**Fig. 2.** The AF Development Methodology

## 2.1 System Behaviour Model

The first model we introduce in this paper is the System Behavior Model (SBM). A system behavior is any distinct set of activities and / or interactions that take place during the operation of the system. For the cloning category, key system behaviors may include agent movement updates, agent registration, and agent de-registration etc.

The SBM is formalized by a customized form of the UML Use Case Diagram. UML Use Case Diagrams is a well-understood approach to describe the functionality of a system in a horizontal way. From an agent-oriented perspective, we adopt the actors as agents that are playing a specific role, and use cases as the behaviors that one associates with these roles [5]. Formally, we introduce two stereotypes to customize those UML Use Case Diagrams: the `<<role>>` stereotype identifies actors that represent roles that agents are playing, and the `<<role-use-case>>` stereotype identifies use cases that occur between agents that are engaged in specific roles.



**Fig. 3.** System Behaviour Model

## 2.2 The Protocol Model

The Protocol Model can be viewed as a formalization of the Interaction Model. It is used to refine the various interaction scenarios into a set of protocols that describe how the agents interact and it encapsulates each of the alternate scenarios associated with a given system behaviour[5]. Each protocol specified in this model is defined as an Agent UML Sequence Diagram [5]. In contrast to the Activity Model, this model concentrates on the interplays between agents.

Tool-based support for protocol creation is provided via the VIPER [11] visual protocol editor. VIPER performs two jobs within this methodology: (1) it supports visual editing of Agent UML Sequence Diagrams, and (2) it automatically generates agent code based upon these diagrams.

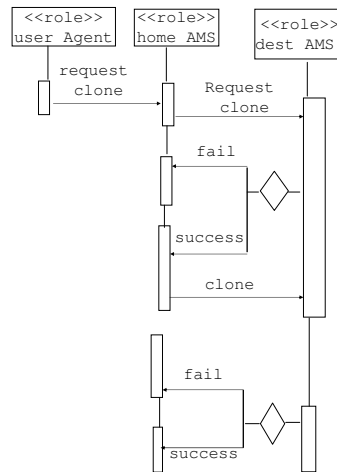


Fig. 4. Protocol Model

### 2.3 The Activity Model

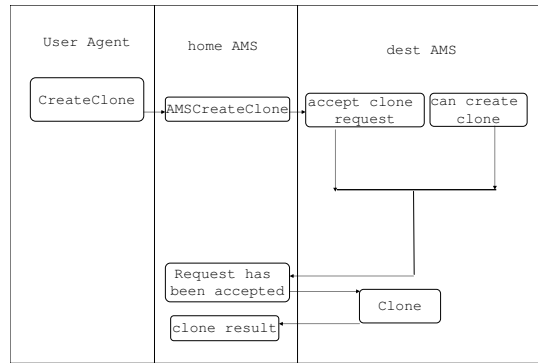
The agent interaction model can sometimes require specifications with very clear processing-thread semantics. The activity diagram expresses operations and the events that trigger them [8]. Compared with the Protocol Model (section 2.2), this model primarily concentrates on those activities that support the system behaviors. We customize the Activity Model through the customization UML Activity Diagrams [2]. Specifically, we customize this diagram through the introduction of a `<<role>>` stereotype, which associates swim lanes with roles [5].

Typically an individual agent can have various activities and similarly a given activity can be performed by more than one agent. On the other hand a role may be viewed as a collection of activities. Thus a *many-to-many* relationship exists between role, activity and agent class.

### 2.4 The Agent Model

The final model we introduce in this paper is the Agent Model. This model is designed to document the various agent types that will be used in the system under development, and the agent classes that will realize these agent types at run-time.

An agent type can be best thought of as an agent role. The agent model within Agent Factory is formalized by way of UML Class Diagram (more details about UML Class Diagram can be found in [12]) that are customized to include:



**Fig. 5.** Activity Model

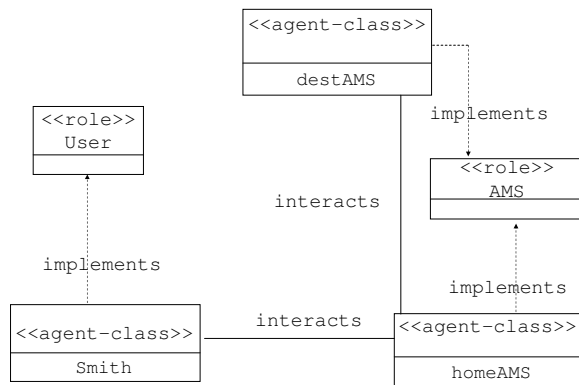
a  $\langle\langle\text{role}\rangle\rangle$  stereotype and an  $\langle\langle\text{agent-class}\rangle\rangle$  stereotype. According to [5] the  $\langle\langle\text{role}\rangle\rangle$  stereotype represents roles, and can be described as a box with two compartments: the first compartment contains the stereotype after which it is the role identifier; while the second compartment contains a list of protocol identifiers. On the other hand, the  $\langle\langle\text{agent-class}\rangle\rangle$  stereotype represents agent classes with the form of a box which includes three compartments: the first compartment contains the stereotype and the agent class identifier, the second compartment contains a list of protocols (not those in the associated roles), and the third compartment contains a set of activity identifiers.

Since an agent implemented by a specific agent class can play more than one role in its life circle and at the same time one role can be played by many agents, in the Agent Factory System we permit a *many-to-many* relationship between  $\langle\langle\text{role}\rangle\rangle$  and  $\langle\langle\text{agent-class}\rangle\rangle$ . That is each agent class can be associated with many roles and each role can be associated with many agent classes.

### 3 Case Study

#### 3.1 Cloning Design

To illustrate our development methodology, we now present the cloning case study. When analyzing this system, two key roles were identified: the User role, which is responsible for requesting cloning, the AMS role, whose duty is to reason for available resources. We formalism this analysis within the System Behavior Model (Figure 3). By way of illustration, we expand upon the 'request clone' and 'Create Clone' system behaviors. Our expansion of this behavior is through the Protocol Model. Figure 5 presents an example of the activity model. Here an agent whose role is user believes that it wants to create a copy of itself. This results in a *CreateClone* event (actually in Agent Factory it is a *CreateClone* belief) that triggers another agent on the same platform with the role of an Agent Management System (AMS) to deal with the request. The AMS agent



**Fig. 6.** Agent Model

then forwards the cloning request to the destination AMS (an AMS agent that is running on a different platform). The destination AMS will accept the request if and only if it has the required capacity to execute the cloning. Once the request is accepted, it will ask the home AMS for the clone materials. The clone result will be sent back to the home AMS after cloning.

To help to clarify how the above interplay works, we further expand upon the ‘request clone’ and ‘Create Clone’ behaviors through the protocol model. Figure 4 presents an example of agent cloning interaction. The rectangle can express individual or sets (i.e., roles or classes) of agents. For example, an individual agent could be labelled ‘<<role>> home AMS’ which means the agent serves as the AMS on the home platform. When it is necessary for the user agent to initiate cloning, the user agent may send a *request\_clone* message to its AMS (home AMS). The home AMS thereafter forwards this request message to the destination AMS (an AMS agent that may have sufficient resources to create the clone and execute the distributed tasks) via a *fipa-inform* message. Upon examining its capacity, the destination AMS will inform the home AMS whether it is able to create the clones or not. Again this information is transformed through a *fipa-inform* message. If the result is positive, the home AMS then can send the entire clone material to the destination AMS. A result will be returned to the home AMS through the *fipa-inform* message indicating the success or failure of the operation.

As the System Behavior, Interaction, and Activity models become more stable (i.e. as a general agreement on how the system behavior emerges), work on these three models stops, and the designer commences work on Agent Model. Specifically, this model presents a view of the roles within the system, and the set of agent classes that will implement those roles (section 2.4). Figure 6 presents the agent model for our cloning mechanism. As can be seen in this model, two

roles are implemented via two agent classes. The Smith class implements the User role while both *homeAMS* class and *destAMS* class implement the AMS role.

### 3.2 Cloning Implementation

To implement the above cloning design, we now introduce the CloneExample Role, an example role that contains two AF roles to (1) initiate the creation of an agent (2) to pass some job to the agent.

```
BELIEF(CreateClone(?name,
?design, ?clonenum, ? task)) &
BELIEF(agentID(?destinationAMS,
?addresses) => COMMIT(Self, Now, BELIEF(true), request(?destinationAMS,
cloneAgent(?name, ?design, ?clonenum, ?task)));
```

**Fig. 7.** CloneExample Rule

```
BELIEF(ToughJob) => COMMIT(Self, Now, BELIEF(true), SEQ(ToughJob, HelloWorld));
```

**Fig. 8.** ToughJob Rule

The rule in Figure 7 states that if the agent believes that it wants to create a specified number of copies (by the parameter ?clonenum) of itself with a specified design (by the parameter ?design). In order to do this, it sends a request to the related AMS to create the agent.

The second rule (Figure 8) delivers a tough job, let us say loop 10,000 times, and a 'prn' job (which is used to print a string 'HelloWord') to the agent. In order to do this it fires the 'ToughJobActuator' actuator and the 'prnActuator' actuator. Note here we use the SEQ operator [4] which means the 'prn' action will not be executed until the 'toughJob' action is completed. In this way, after cloning the 'prn' task can be delivered to the cloned agent.

Here if an agent believes that the current commitment is a tough job (which will cause the system overloading) it will take another belief that it needs to

create a specified number of copies of itself to release the system overloading (Figure 7). This belief leads to a *fipa-request* message which requires the AMS agent to clone (Figure 7). During the clone process the commitment with the SEQ key word (Figure 8) is separated into some sub-tasks which will be re-allocated to the cloned agents. In this example, the HelloWorld sub-task will be delivered to the cloned agent after cloning.

## 4 Discussion and Conclusion

The methodology presented in this paper represents one of a number of potential approaches to fabricating MAS. It is not possible in a paper to do justice to the large amount of work being done.

Gaia is a methodology that supports the analysis and design phases. It is applicable to a wide range of agent systems. Furthermore it can deal with both macro-level (societal) and micro-level (agent) aspects of systems [9]. Gaia takes the view that a system can be seen as a society or an organization of agents. The methodology is applied after the requirements are gathered and specified, and covers the analysis and design phases. However according to [14] the disadvantages of Gaia is that (1) Gaia does not directly deal with particular modeling techniques. (2) Gaia does not directly deal with implementation issues and (3) Gaia does not explicitly deal with the activities of the requirements capturing and modeling, and specifically of early requirements engineering.

Tropos [6] proposes both a software development methodology and a development framework which are founded on concepts used to model early requirements and complements proposals for agent-oriented programming platforms. Compared with AFDM, Tropos provides an early requirements phase, which the AF Methodology does not. However tool support for Tropos is currently only in the form of diagram editor[8], rather than the automatically generation of agent code that is part of VIPER.

Recently a cooperation has been established between the Foundation of Intelligent Physical Agents (FIPA) and the Object Management Group (OMG). As a result of this cooperation, the framework of Agent UML (AUML) was introduced [1]. The AUML approach introduces the Agent Interaction Protocol (AIP) for agent communication and constraints on messages.

In summary, this paper presents the key aspects of the Agent Factory Development Methodology that is founded upon the industry standard UML design notation. This methodology has been employed in the development of a number of large scale agent-based applications. Moreover the models underpinning this methodology perform two jobs i.e.design reuse and automated partial code generation. On the other hand agent cloning is considered as an approach to release the system bottlenecks. Finally, we illustrate our AFDM through a trial example namely design and implementation of agent cloning in the AF System.

## References

1. B. Bauer. Extending uml for the specification of interaction protocols. Technical report, submission for the 6th Call for Proposal of FIPA and revised version part of FIPA 99, U. C. Berkeley, April 1999.
2. James Rumbaugh Booch, Grady and Ivar Jacobson. *The Unified Language User Guide*. Addison-Wesley, Reading, MA, 1999.
3. D.Phelan R.Strahan and R.Collier C.Muldoon, G.M.P.O'Hare. *Access:An agent architecture for ubiquitous service delivery*, volume LNAI 2782. Springer Verlag, cooperative information agents vii edition, 2003.
4. Rem William Collier. Agent factory: A framework for the engineering of agent-oriented applications, 2001.
5. O'Hare G.M.P. Collier R., Rooney C. A uml-based software engineering methodology for agent factory. Banff, Alberta, Canada, June 2004. Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering (SEKE-2004).
6. F.Giunchiglia, J Mylopoulos, and A. Perini. The tropos software development methodology: Process, models and diagrams. Technical report, Technical Report DIT-02-008, Informatica e Telecomunicazioni, Universita degli Studi di Trento, 2001, 2001.
7. G.M.P.O'Hare and M.O'Grady. Gulliver's genie:a multi-agent system for ubiquitous and intelligent content delivery. *Computer Communication*, 26(11):1178-1187, 2003.
8. H. V. D. Parunack J. Odell and B. Bauer. Extending uml for agents. In Proceedings of AOIS Workshop at AAAI 2000, 2000.
9. N. R. Jennings M. Wooldridge and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 1999.
10. Collier R., Rooney C.F.B., O'Donoghue R.P.S., and O'Hare G.M.P. Mobile bdi agents. *11th Irish Conference*, 2000.
11. C. F. B. Rooney, R. W. Collier, and G. M. P. O'Hare. Viper: Visual protocol editor. In Proceedings of COORDINATION 2004, Pisa, Italy, 2004.
12. Ivar Jacobson Rumbaugh, James and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, MA, 1999.
13. G.O'Hare T.Lowen and P.O'Hare. Mobile agents point the way:context sensitive service delivery through mobile lightweight agents. Bologna,Italy, 2002. Proceedings of First International Joint Conference on Autonomous Agents and Multi-Agent Systems Conference(AAMAS2002), AAAI Publisher.
14. Franco Zambonelli, Nicholas R. Jenningsy, and Michael Wooldridgez. Developing multiagent systems: The gaia methodology, dipartimento di scienze e metodi dell'ingegneria. October 2003.