

# Modeling-Notation Source: GAIA

Version 03-03-12 17:25EST  
Document author- Alfredo Garro

Gaia is a Methodology which has been specifically tailored to the analysis and design of agent-based systems. Before giving an overview of Gaia Methodology it is worth commenting on the characteristics of domains for which Gaia is appropriate. Gaia is appropriate for the development of systems with the following main characteristics:

- Agents are coarse-grained computational systems, each making use of significant computational resources (think of each agent as having the resources of a Unix process).
- It is assumed that the goal is to obtain a system that maximises some global quality measure, but which may be sub-optimal from the point of view of the system components. Gaia is *not* intended for systems that admit the possibility of true conflict.
- Agents are heterogeneous, in that different agents may be implemented using different programming languages, architectures, and techniques. We make no assumptions about the delivery platform;
- The organisation structure of the system is static, in that inter-agent relationships do not change at run-time.
- The abilities of agents and the services they provide are static, in that they do not change at run-time.
- The overall system contains a comparatively small number of different agent types (less than 100).

## 1 GAIA's Model

Gaia is intended to allow an analyst to go systematically from a statement of requirements to a design that is sufficiently detailed that it can be implemented directly. Analysis and design can be thought of as a process of developing increasingly detailed *models* of the system to be constructed (Figure 1).

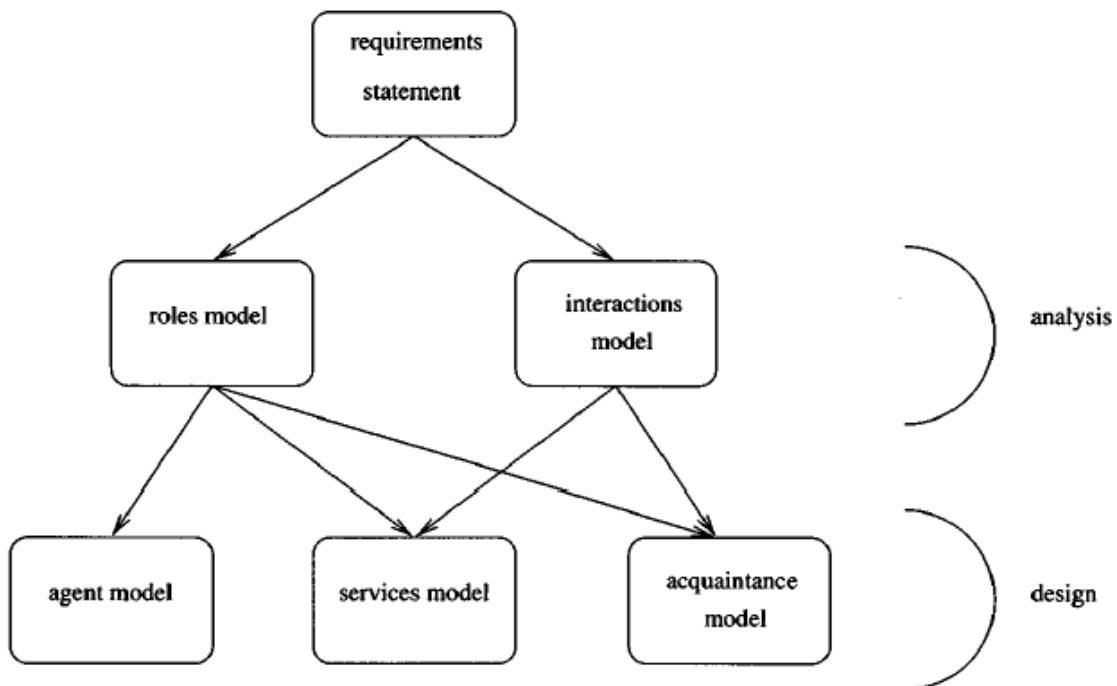


Figure 1. GAIA's Model

33  
34  
35  
36  
37  
38  
39  
40  
41

GAIA provides an agent-specific set of **concepts** through which a software engineer can understand and model a complex system. In particular, Gaia encourages a developer to think of building agent-based systems as a process of *organisational design*. The main Gaian concepts can be divided into two categories: *abstract* and *concrete* (Table 1). Abstract entities are those used during **analysis** to conceptualise the system, but which do not necessarily have any *direct* realisation within the system. Concrete entities, in contrast, are used within the **design** process, and will typically have direct counterparts in the run-time system.

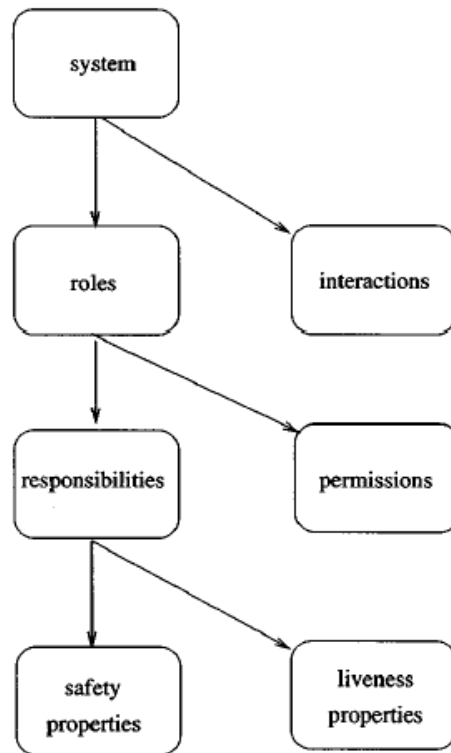
Abstract concepts	Concrete concepts
Roles	Agent Types
Permissions	Services
Responsibilities	Acquaintances
Protocols	
Activities	
Liveness properties	
Safety properties	

42  
43  
44  
45

Table 1. Abstract and concrete concepts in Gaia

## 2 Analysis

47 The objective of the analysis stage is to develop an understanding of the system and its structure (without reference to any implementation detail). This understanding is captured in the system's *organisation*. An organisation can be seen as a collection of roles, that stand in certain relationships to one another, and that take part in systematic, institutionalised patterns of interactions with other roles (Figure 2).



52  
53

Figure 2. Analysis Concepts



10 intended to keep filled), and the *coffeeStatus* (that indicates whether the machine is full or empty). In addition, the  
 11 role has permission to change the value *coffeeStock*.  
 12

Role Schema:	<i>name of role</i>
Description	<i>short English description of the role</i>
Protocols and Activities	<i>protocols and activities in which the role plays a part</i>
Permissions	<i>“rights” associated with the role</i>
Responsibilities	
Liveness	<i>liveness responsibilities</i>
Safety	<i>safety responsibilities</i>

13  
 14  
 15

Figure 3. Template for role schemata.

Role Schema: COFFEEFILLER	
Description: This role involves ensuring that the coffee pot is kept filled, and informing the workers when fresh coffee has been brewed.	
Protocols and Activities: Fill, InformWorkers, <u>CheckStock</u> , AwaitEmpty	
Permissions:	
reads	supplied <i>coffeeMaker</i> // name of coffee maker <i>coffeeStatus</i> // full or empty
changes	<i>coffeeStock</i> // stock level of coffee
Responsibilities	
Liveness: COFFEEFILLER = (Fill. InformWorkers. <u>CheckStock</u> . AwaitEmpty) <sup>ω</sup>	
safety:	
•	<i>coffeeStock</i> > 0

16  
 17  
 18

Figure 4. Schema for role CoffeeFiller

19 **2.2 The interaction model**

20 In Gaia links between roles are represented in the *interaction model*. This model consists of a set of *protocol*  
 21 *definitions*, one for each type of inter-role interaction. Here a protocol can be viewed as an institutionalised pattern  
 22 of interaction. That is, a pattern of interaction that has been formally defined and abstracted away from any  
 23 particular sequence of execution steps. Viewing interactions in this way means that attention is focused on the  
 24 essential nature and purpose of the interaction, rather than on the precise ordering of particular message  
 25 exchanges (cf. the interaction diagrams of OBJECTORY [2, pp. 198-203] or the scenarios of FUSION [2]).

26 A protocol definition consists of the following attributes:

- 27 ▪ *purpose*: brief textual description of the nature of the interaction (e.g., “information request”, “schedule activity” and “assign task”);
- 28 ▪ *initiator*: the role(s) responsible for starting the interaction;
- 29 ▪ *responder*: the role(s) with which the initiator interacts;
- 30 ▪ *inputs*: information used by the role initiator while enacting the protocol;
- 31 ▪ *outputs*: information supplied by/to the protocol responder during the course of the interaction;
- 32 ▪ *processing*: brief textual description of any processing the protocol initiator performs during the course of the interaction.

33  
 34  
 35  
 36

## 37 2.3 The Analysis Process

38  
39 The analysis stage of Gaia can now be summarised:

- 40 1. Identify the *roles* in the system. Roles in a system will typically correspond to:
- 41     ▪ individuals, either within an organisation or acting independently;
  - 42     ▪ departments within an organisation; or
  - 43     ▪ organisations themselves.

44     *Output:* A prototypical roles model—a list of the key roles that occur in the system, each with an informal, unelaborated description.

- 45  
46 2. For each role, identify and document the associated *protocols*. Protocols are the patterns of interaction that occur in the system between the various roles. For example, a protocol may correspond to an agent in the role of Buyer submitting a bid to another agent in the role of Seller.

47     *Output:* An interaction model, which captures the recurring patterns of inter-role interaction.

- 48  
49 3. Using the protocol model as a basis, elaborate the roles model.

50     *Output:* A fully elaborated roles model, which documents the key roles occurring in the system, their permissions and responsibilities, together with the protocols and activities in which they participate.

- 51  
52 4. Iterate stages (1)-(3).  
53  
54  
55

## 56 3 Design

57  
58 The aim in Gaia is to transform the analysis models into a sufficiently low level of abstraction that traditional design techniques (including object-oriented techniques) may be applied in order to implement agents. To put it another way, Gaia is concerned with how a society of agents cooperate to realise the system-level goals, and what is required of each individual agent in order to do this. Actually *how* an agent realises its services is beyond the scope of Gaia, and will depend on the particular application domain.

60 The Gaia design process involves generating three models (see Figure 1). The **agent model** identifies the *agent types* that will make up the system, and the *agent instances* that will be instantiated from these types. The **services model** identifies the main services that are required to realise the agent's role. Finally, the **acquaintance model** documents the lines of communication between the different agents.

61  
62  
63  
64  
65  
66  
67

### 68 3.1 The Agent Model

69 The purpose of the Gaia agent model is to document the various *agent types* that will be used in the system under development, and the *agent instances* that will realise these agent types at run-time.

70  
71

72 An agent type is best thought of as a set of agent roles. It is defined using a simple *agent type tree*, in which leaf nodes correspond to roles, (as defined in the roles model), and other nodes correspond to agent types. If an agent type *t1* has children *t2* and *t3*, then this means that *t1* is composed of the roles that make up *t2* and *t3*.

73  
74  
75

### 76 3.2 The services model

77 As its name suggests, the aim of the Gaia services model is to identify the *services* associated with each agent role, and to specify the main properties of these services.

78 By a service, GAIA mean a *function* of the agent. A service is a coherent block of activity in which an agent will engage. It should be clear there every activity identified at the analysis stage will correspond to a service, though not every service will correspond to an activity.

81 For each service that may be performed by an agent, it is necessary to document its properties. Specifically, we must identify the *inputs*, *outputs*, *pre-conditions*, and *post-conditions* of each service. Inputs and outputs to services will be derived in an obvious way from the protocols model. Pre- and post-conditions represent constraints on services. These are derived from the safety properties of a role. Note that by definition, each role will be associated with at least one service. The services that an agent will perform are derived from the list of protocols, activities, responsibilities and the liveness properties of a role. The Gaia services model does *not* prescribe an implementation for the services it documents. The developer is free to realise the services in any implementation framework deemed appropriate. For example, it may be decided to implement services directly as methods in an object-oriented language. Alternatively, a service may be decomposed into a number of methods.

82  
83  
84  
85  
86  
87  
88  
89  
90  
91

### 92 3.3 The acquaintance model

93 The final Gaia design model is probably the simplest: the *acquaintance model*.

94 Acquaintance models simply define the communication links that exist between agent types. They do *not* define  
95 what messages are sent or when messages are sent, they simply indicate that communication pathways exist. In  
96 particular, the purpose of an acquaintance model is to identify any potential communication bottlenecks, which  
97 may cause problems at run-time

98 An agent acquaintance model is simply a graph, with nodes in the graph corresponding to agent types and arcs  
99 in the graph corresponding to communication pathways. Agent acquaintance models are *directed* graphs, and so  
00 an arc  $a \rightarrow b$  indicates that a will send messages to b, but not necessarily that b will send messages to a. An  
01 acquaintance model may be derived in a straightforward way from the roles, protocols, and agent models.  
02

### 03 3.4 The design process

04 The Gaia design stage can now be summarised:

05 1. Create an *agent model*:

- 06     ▪ aggregate roles into *agent types*, and refine to form an agent type hierarchy;
- 07     ▪ document the instances of each agent type using instance annotations.

08 2. Develop a *services model*, by examining activities, protocols, and safety and liveness properties of roles.

09 3. Develop an *acquaintance model* from the interaction model and agent model.  
10  
11  
12

## 13 4 References

14  
15 [1] M. Wooldrige, N. R. Jennings and D. Kinny, *The Gaia Methodology for Agent-Oriented Analysis and Design*,  
16 Autonomous Agents and Multi-Agent Systems, volume 3, pp 285-312, Kluwer Academic Publishers, The  
17 Netherlands, 2000.  
18

19 [2] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes, *Object-Oriented*  
20 *Development: The fusion Method*. Prentice Hall International: Hemel Hempstead, England, 1994.  
21  
22  
23  
24  
25  
26  
27  
28  
29

30

31

32

33

34

35

36

37

38

39 **5**