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](image-url)
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.

<table>
<thead>
<tr>
<th>Abstract concepts</th>
<th>Concrete concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roles</td>
<td>Agent Types</td>
</tr>
<tr>
<td>Permissions</td>
<td>Services</td>
</tr>
<tr>
<td>Responsibilities</td>
<td>Acquaintances</td>
</tr>
<tr>
<td>Protocols</td>
<td></td>
</tr>
<tr>
<td>Activities</td>
<td></td>
</tr>
<tr>
<td>Liveness properties</td>
<td></td>
</tr>
<tr>
<td>Safety properties</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Abstract and concrete concepts in Gaia

2 Analysis

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).

Figure 2. Analysis Concepts
The most abstract entity is the **system** with the meaning of **society or organisation**. The idea of a system as a society is useful when thinking about the next level in the concept hierarchy: **roles**.

A **role** is defined by four attributes: **responsibilities, permissions, activities, and protocols**. Responsibilities determine functionality and, as such, are perhaps the key attribute associated with a role. An example responsibility associated with the role of company president might be calling the shareholders meeting every year.

Responsibilities are divided into two types: **liveness properties** and **safety properties**. Liveness properties intuitively state that something good happens. They describe those states of affairs that an agent must bring about, given certain environmental conditions. In contrast, safety properties are **invariants**. Intuitively, a safety property states that nothing bad happens (i.e., that an acceptable state of affairs is maintained across all states of execution). In order to realise responsibilities, a role has a set of **permissions**. Permissions are the rights associated with a role. The permissions of a role thus identify the resources that are available to that role in order to realise its responsibilities. Permissions tend to be **information resources**. For example, a role might have associated with it the ability to read a particular item of information, or to modify another piece of information. A role can also have the ability to **generate** information.

The **activities** of a role are computations associated with the role that may be carried out by the agent without interacting with other agents. Finally, a role is also identified with a number of **protocols**, which define the way that it can interact with other roles. For example, a seller role might have the protocols Dutch auction and English auction associated with it; the Contract Net Protocol is associated with the roles manager and contractor.

Thus, the organisation model in Gaia is comprised of two further models: the **roles model** and the **interaction model**.

### 2.1 The roles model

The roles model identifies the key roles in the system. Here a role can be viewed as an abstract description of an entity's expected function. Such roles (or offices) are characterised by two types of attribute:

- **The permissions/rights associated with the role.** A role will have associated with it certain permissions, relating to the type and the amount of resources that can be exploited when carrying out the role. In our case, these aspects are captured in an attribute known as the role's **permissions**.

- **The responsibilities of the role.** A role is created in order to do something. That is, a role has a certain functionality. This functionality is represented by an attribute known as the role's **responsibilities**.

#### 2.1.1 Permissions

The **permissions** associated with a role have two aspects:

- they identify the resources that can legitimately be used to carry out the role intuitively, they say what can be spent while carrying out the role;
- they state the resource limits within which the role executor must operate intuitively, they say what can’t be spent while carrying out the role.

Gaia makes use of a formal notation for expressing permissions that is based on the fusion notation for operation schemata [2, pp. 26-31].

#### 2.1.2 Responsibilities

The **functionality** of a role is defined by its **responsibilities**. These responsibilities can be divided into two categories: **liveness** and **safety** responsibilities. Liveness responsibilities are those that, intuitively, state that something good happens. Liveness responsibilities are so called because they tend to say that something will be done, and hence that the agent carrying out the role is still alive. Liveness responsibilities tend to follow certain patterns. For example, the **guaranteed response** type of achievement goal has the form "a request is always followed by a response". The **infinite repetition** achievement goal has the form "x will happen infinitely often". In Gaia, liveness properties are specified via a **liveness expression**, which defines the "life-cycle" of the role. Liveness expressions are similar to the **life-cycle** expression of fusion [2], which are in turn essentially regular expressions.

It is now possible to precisely define the Gaia roles model. A roles model is comprised of a set of **role schemata**, one for each role in the system. A role schema draws together the various attributes discussed above into a single place (Figure 3). An exemplar instantiation is given for the CoffeeFiller role in Figure 4. This schema indicates that CoffeeFiller has permission to read the **coffeeMaker** parameter (that indicates which coffee machine the role is
intended to keep filled), and the coffeeStatus (that indicates whether the machine is full or empty). In addition, the role has permission to change the value coffeeStock.

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

Figure 3. Template for role schemata.

<table>
<thead>
<tr>
<th>Role Schema: COFFEE_FILLER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: This role involves ensuring that the coffee pot is kept filled, and informing the workers when fresh coffee has been brewed.</td>
</tr>
<tr>
<td>Protocols and Activities: Fill, InformWorkers, CheckStock, AwaitEmpty</td>
</tr>
<tr>
<td>Permissions: reads supplied coffeeMaker // name of coffee maker</td>
</tr>
<tr>
<td>changes coffeeStatus // full or empty</td>
</tr>
<tr>
<td>coffeeStock // stock level of coffee</td>
</tr>
<tr>
<td>Responsibilities</td>
</tr>
<tr>
<td>Liveness:</td>
</tr>
<tr>
<td>safety:</td>
</tr>
<tr>
<td>• coffeeStock &gt; 0</td>
</tr>
</tbody>
</table>

Figure 4. Schema for role CoffeeFiller

2.2 The interaction model

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

A protocol definition consists of the following attributes:

- **purpose**: brief textual description of the nature of the interaction (e.g., "information request", "schedule activity" and "assign task");
- **initiator**: the role(s) responsible for starting the interaction;
- **responder**: the role(s) with which the initiator interacts;
- **inputs**: information used by the role initiator while enacting the protocol;
- **outputs**: information supplied by/to the protocol responder during the course of the interaction;
- **processing**: brief textual description of any processing the protocol initiator performs during the course of the interaction.
2.3 The Analysis Process

The analysis stage of Gaia can now be summarised:

1. Identify the roles in the system. Roles in a system will typically correspond to:
   - individuals, either within an organisation or acting independently;
   - departments within an organisation; or
   - organisations themselves.
   
   Output: A prototypical roles model—a list of the key roles that occur in the system, each with an informal, unelaborated description.

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

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

4. Iterate stages (1)-(3).

3 Design

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.

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.

3.1 The Agent Model

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.

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.

3.2 The services model

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.

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 that every activity identified at the analysis stage will correspond to a service, though not every service will correspond to an activity.

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.
3.3 The acquaintance model

The final Gaia design model is probably the simplest: the acquaintance model. Acquaintance models simply define the communication links that exist between agent types. They do not define what messages are sent or when messages are sent, they simply indicate that communication pathways exist. In particular, the purpose of an acquaintance model is to identify any potential communication bottlenecks, which may cause problems at run-time. An agent acquaintance model is simply a graph, with nodes in the graph corresponding to agent types and arcs in the graph corresponding to communication pathways. Agent acquaintance models are directed graphs, and so an arc \( a \rightarrow b \) indicates that \( a \) will send messages to \( b \), but not necessarily that \( b \) will send messages to \( a \). An acquaintance model may be derived in a straightforward way from the roles, protocols, and agent models.

3.4 The design process

The Gaia design stage can now be summarised:

1. Create an agent model:
   - aggregate roles into agent types, and refine to form an agent type hierarchy;
   - document the instances of each agent type using instance annotations.
2. Develop a services model, by examining activities, protocols, and safety and liveness properties of roles.
3. Develop an acquaintance model from the interaction model and agent model.

4 References