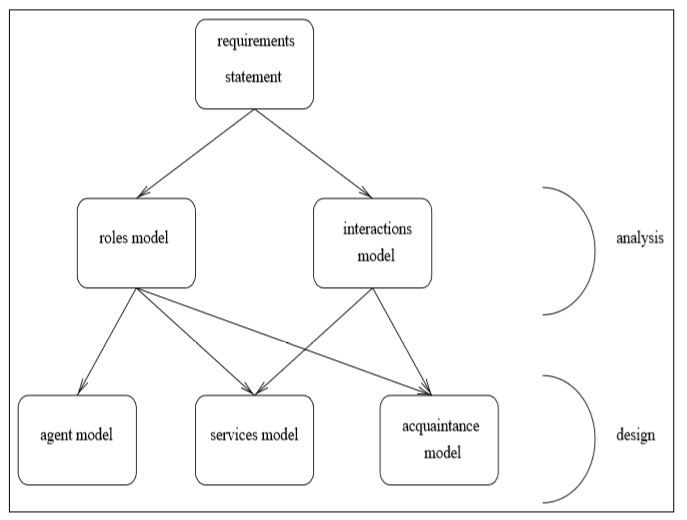
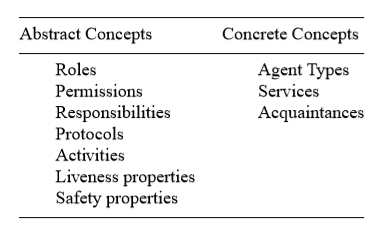
# The Gaia Methodology for Agent-Oriented Analysis and Design

Agents may be used by software developers to more naturally understand, model, and develop an important class of complex distributed systems. If agents are to realize their potential as a software engineering paradigm, then it is necessary to develop software engineering techniques that are speciﬁcally tailored to them. Existing software development techniques (for example, object-oriented analysis and design) are unsuitable for this task. There is a fundamental mismatch between the concepts used by object-oriented developers and the agent-oriented view. In particular, extant approaches fail to adequately capture an agent’s ﬂexible, autonomous problem-solving behavior, the richness of an agent’s interactions, and the complexity of an agent system’s organizational structures. For these reasons, we study a methodology called **Gaia**, which has been speciﬁcally tailored to the analysis and design of agent-based systems.

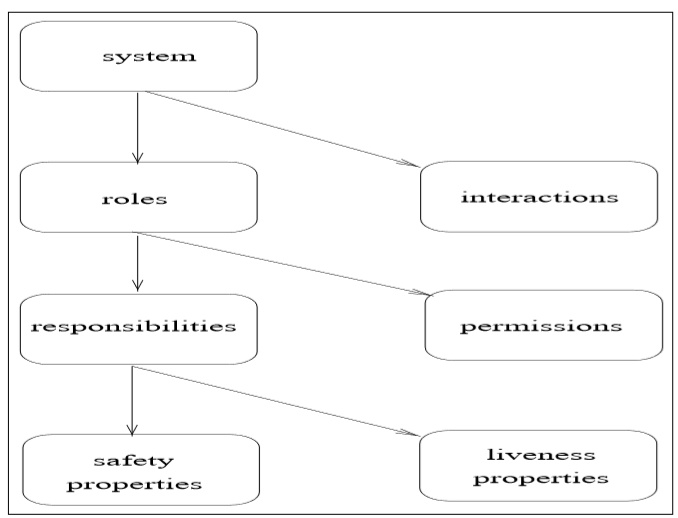
## A conceptual framework

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. In applying Gaia, the analyst moves from abstract to increasingly concrete concepts. Analysis and design can be thought of as a process of developing increasingly detailed models of the system to be constructed.

The main Gaian concepts can be divided into two categories: abstract and concrete.

**Abstract** entities are those used during **analysis** to conceptualize the system, but which do not necessarily have any direct realization within the system. **Concrete** entities, in contrast, are used within the **design** process, and will typically have direct counterparts in the run-time system.

## Analysis

The objective of the analysis stage is to develop an understanding of the system and its structure. In our case, this understanding is captured in the **system’s organization**. We view an organization as a **collection of roles**, that stand in certain relationships to one another, and that take part in systematic, institutionalized patterns of interactions with other roles.

We can think of a **system** as a society. This is useful when thinking about the next level in the concept hierarchy: roles. It is not unusual for one individual to take on many roles. Conversely, there may be many individuals that take on a single role.

A **role** is deﬁned by four attributes: responsibilities, permissions, activities, and protocols.

1. **Responsibilities** determine functionality

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 condition.
* **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). An example might be “ensure the reactor temperature always remains in the range 0-100”

1. In order to realize 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 realize its responsibilities. 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.
2. The **activities** of a role are computations associated with the role that may be carried out by the agent without interacting with other agents. Activities are thus “private” actions.
3. **Protocols** deﬁne the way that the role can interact with other roles. For example, a “seller” role might have the protocols “Dutch auction” and “English auction” associated with it.

The organization model in Gaia is comprised of two further models: **the roles model** and **the interaction model**.

### 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**. Roles are characterized by two types of attribute:

1. 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.

1. The **responsibilities** of the role.

A role is created in order to do something. That is, a role has a certain functionality.

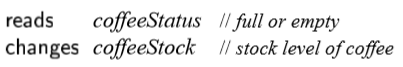
#### Permissions

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.

In Gaia, we think of resources as relating only to the information or knowledge the agent has. That is, in order to carry out a role, an agent will typically be able to access certain information.

Gaia makes use of a formal notation for expressing permissions that is based on the FUSION notation for operation schemata. To introduce our concepts, we will use the example of a COFFEEFILLER role (the purpose of this role is to ensure that a coffee pot is kept full of coffee for a group of workers). The following is a simple illustration of the permissions associated with the role COFFEEFILLER:



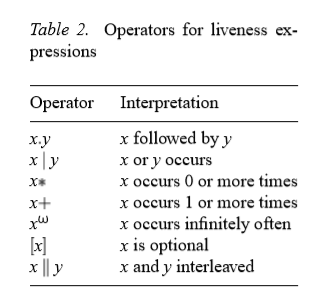
This speciﬁcation deﬁnes two permissions for COFFEEFILLER: it says that the agent carrying out the role has permission to access the value coffeeStatus, and has permission to both read and modify the value coffeeStock. There is also a third type of permission, **generates**, which indicates that the role is the producer of a resource (not shown in the example). Note that these permissions relate to knowledge that the agent has. That is, coffeeStatus is a representation on the part of the agent of some value in the real world.

Some roles are parameterized by certain values. For example, we can generalize the COFFEEFILLER role by parameterizing it with the coffee machine that is to be kept reﬁlled. This is speciﬁed in a permissions deﬁnition by the **supplied** keyword, as follows:

#### 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 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”.



The general form of a liveness expression is:

Ritaglio schermata

where ROLENAME is the name of the role whose liveness properties are being deﬁned, and expression is the liveness expression deﬁning the liveness properties of ROLENAME.

The atomic components of a liveness expression are either **activities** or **protocols**.

* An **activity** corresponds to a unit of action that the agent may perform, which does not involve interaction with any other agent. Activity names are underlined.
* **Protocols** are activities that do require interaction with other agents.

To illustrate liveness expressions, consider again the above-mentioned responsibilities of the COFFEEFILLER role:

Ritaglio schermata

This expression says that COFFEEFILLER consists of executing the protocol Fill, followed by the protocol InformWorkers, followed by the activity CheckStock and the protocol AwaitEmpty. The sequential execution of these protocols and activities is then repeated inﬁnitely often.

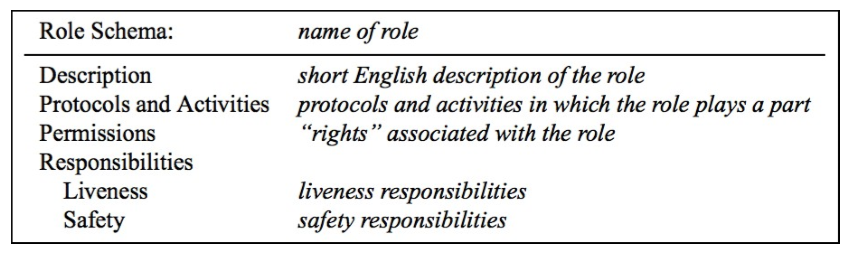
Complex liveness expressions can be made easier to read by structuring them. A simple example illustrates how this is done:

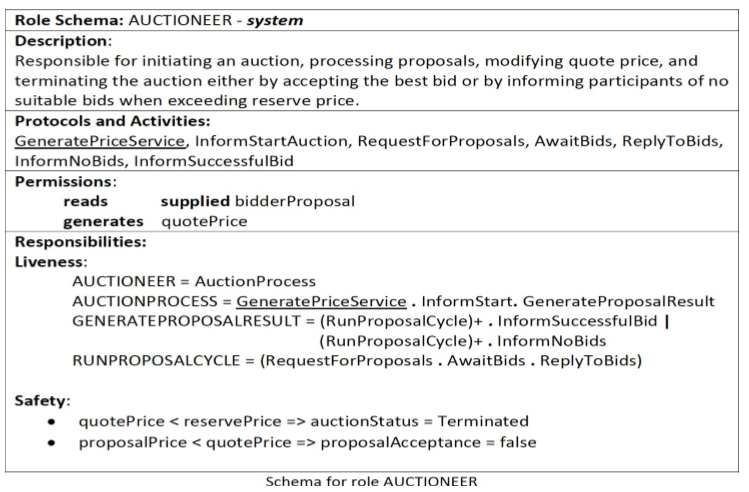


* **Safety requirements** in Gaia are speciﬁed by means of a list of **predicates**. These predicates are typically expressed over the variables listed in a role’s permissions attribute. Returning to our COFFEEFILLER role, an agent carrying out this role will generally be required to ensure that the coffee stock is never empty. We can do this by means of the following safety expression:

Ritaglio schermata

By convention, we simply list safety expressions as a bulleted list, each item in the list expressing an individual safety responsibility. It is implicitly assumed that these responsibilities apply across all states of the system execution. If the role is of inﬁnitely long duration (as in the COFFEEFILLER example), then the invariants must always be true.





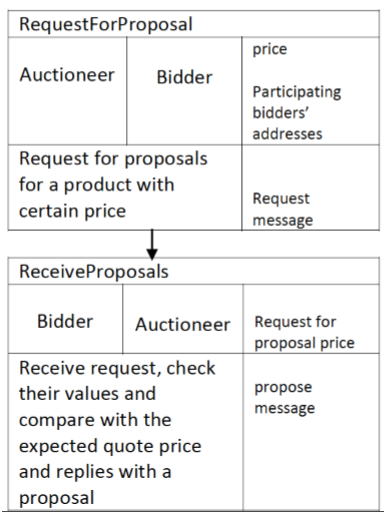
### The interaction model

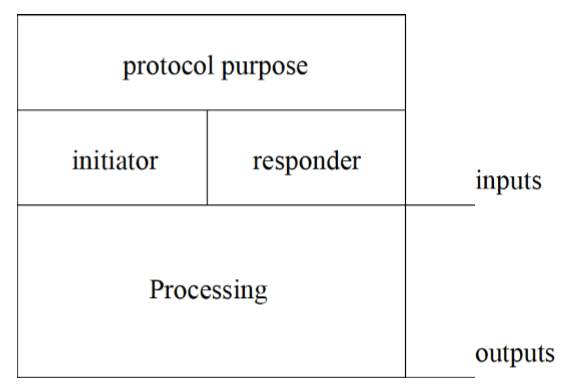
There are inevitably dependencies and relationships between the various roles in a multiagent organization. In Gaia, such links between roles are represented in the interaction model.

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 institutionalized pattern of interaction. This approach means that a single protocol deﬁnition will typically give rise to a number of message interchanges in the run time system.

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.





### The Analysis Process

The analysis stage of Gaia can now be summarized:

1. Identify the **roles** in the system. Roles in a system will typically correspond to:

* individuals, either within an organization or acting independently;
* departments within an organization; or
* organizations themselves.

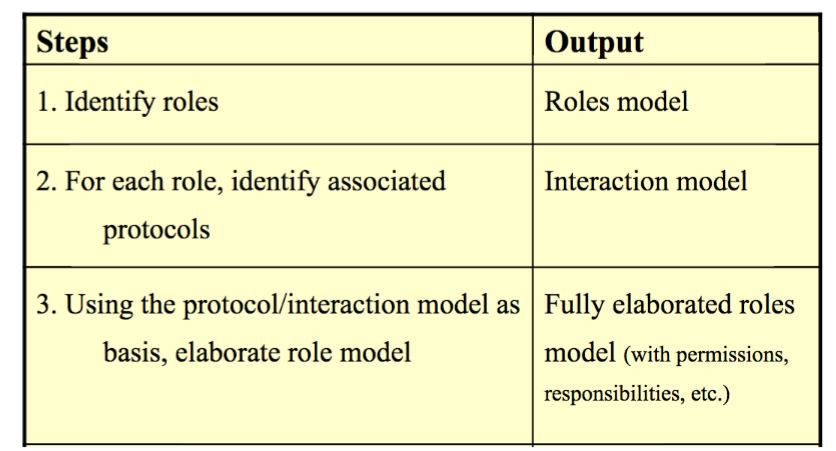
🡪 ***Output***: A prototypical roles model — a list of the key roles that occur in the system, each with an informal, unelaborated description.

1. 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.

1. 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.



## Design

The Gaia design process involves generating three models:

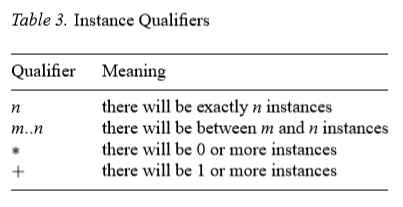
1. The **agent model** identiﬁes the *agent types* that will make up the system, and the *agent instances* that will be instantiated from these types.
2. The **services model** identiﬁes the main services that are required to realize the agent’s role.
3. The **acquaintance model** documents the lines of communication between the different agents.

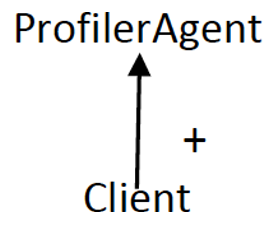
### The agent model

An **agent type** is best thought of as a set of agent roles. There may in fact be a one-to-one correspondence between roles and agent types. However, this need not be the case. Efﬁciency will also be a major concern at this stage: a designer will almost certainly want to optimize the design, and one way of doing this is to aggregate a number of agent roles into a single type.

The agent model 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.

We document the agent instances that will appear in a system by annotating agent types in the agent model. An annotation n means that there will be exactly n agents of this type in the run-time system. An annotation *m…n* means that there will be no less than m and no more than n instances of this type in a run-time system (m<=n). An annotation \* means that there will be zero or more instances at run-time, and + means that there will be one or more instances at run-time.





### The Service 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, we mean a **function** of the agent.

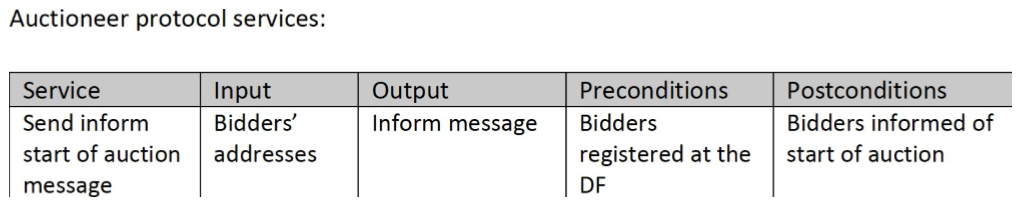
In OO terms, a service would correspond to a *method*; however, we do not mean that services are available for other agents in the same way that an object’s methods are available for another object to invoke. Rather, a service is simply a single, coherent **block of activity** in which an agent will 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. We must identify the **inputs**, **outputs**, **pre-conditions**, and **postconditions** 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.

*The services that an agent will perform are derived from the list of protocols, activities, responsibilities and the liveness properties of a role*.

For example, returning to the coffee example, there are four activities and protocols associated with this role: Fill, InformWorkers, CheckStock, and AwaitEmpty. In general, there will be at least one service associated with each protocol. In the case of CheckStock, for example, the service (which may have the same name), will take as input the stock level and some threshold value, and will simply compare the two. The pre- and post-conditions will both state that the coffee stock level is greater than 0. This is one of the safety properties of the role COFFEEFILLER.



### 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 runtime.

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.

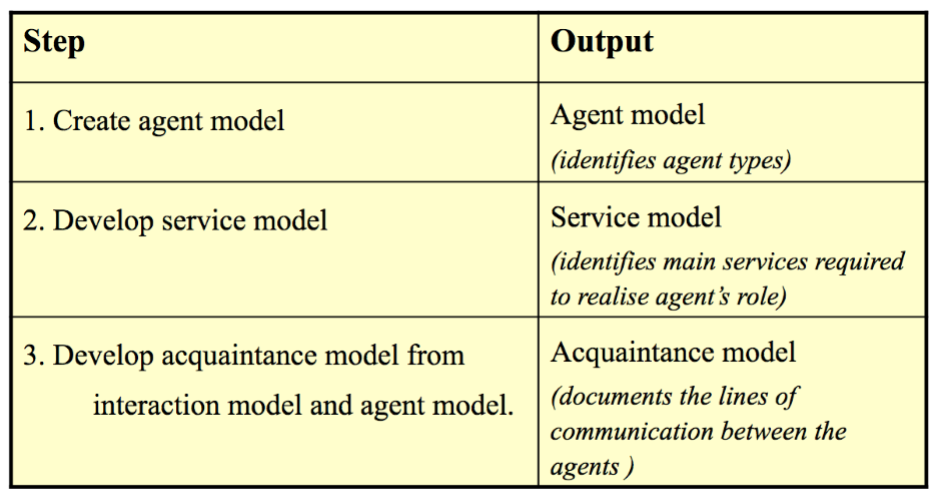
### The Design Process

The Gaia design stage can now be summarized:

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.

1. Develop a **services model**, by examining activities, protocols, and safety and liveness properties of roles.
2. Develop an **acquaintance model** from the interaction model and agent model.



# From m-GAIA ro GRASSHOPPER: engineering mobile agent applications

m-GAIA is an extension to the GAIA methodology for modelling **mobile agent systems**. m-GAIA incorporates explicit constructs to perform the analysis and design of multiagent systems which include mobile agents.

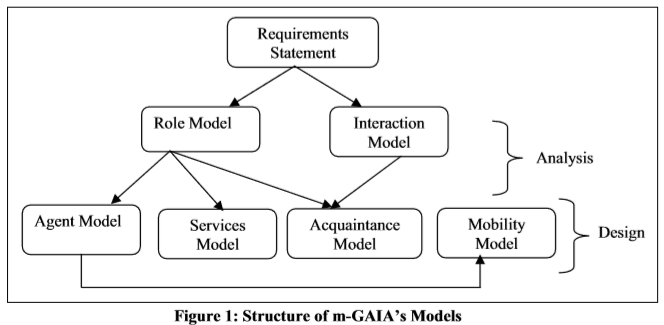
## m-GAIA

The GAIA methodology allows the software developer to analyze and design the system after the requirements are collected and move from abstract (analysis) to concrete (design) level of agent systems. The objective of the *analysis phase* is to obtain an understanding of the system and its structure. The analysis phase consists of the roles model and the interaction model. The roles model identifies the roles in the system and the interaction model identifies the interactions between the roles found. There are four attributes of roles: responsibilities, permissions, activities, and protocols.

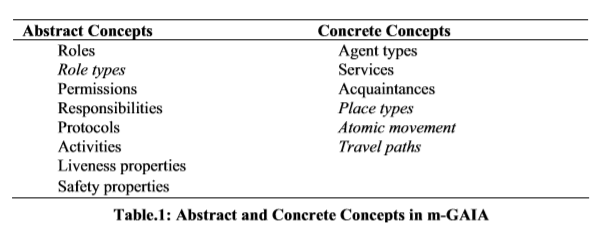
* Responsibilities consist of two properties: liveness property defines the continual execution of the role within the system; safety property is a condition that must be maintained to avoid system behavior that is contrary to its system requirements.
* Permissions define the access privileges or rights of roles.
* Activities are tasks that need to be performed by roles without interaction with other roles.
* Protocols are activities that involve interactions with other roles.

The objectives of the *design phase* are to convert the system from an abstract level to a concrete level and to ease implementation. The design phase consists of the agent model, the services model, and the acquaintance model. The agent model is used to map the roles to agent types. The services model lists the services that each role can provide and be associated with. The acquaintance model defines the relationships between agents.

In order to support conceptual modelling of mobile multiagent systems, m-GAIA incorporates the existing models of GAIA and adds a new model, namely, the **mobility model**. m-GAIA still consists of two phases, which are the analysis and design phases. The analysis phase includes the roles model and the interaction model. The design phase includes the agent model, the acquaintance model, the services model, and the *mobility model*.



The mobility model of m-GAIA defines the mobility characteristics of agents further, such as identifying the movements and travel path of each mobile agent. m-GAIA’s models still serve the same purpose as the corresponding models in GAIA.



It must be noted that in Table 1, the italicized concepts are unique to m-GAIA and mainly aim to support modelling agent mobility in multiagent system. The additional features involve modifications to two of the existing GAIA’s models, which occur in the roles model and the agent model: (1) *In the roles model*, the roles identified are categorized into three distinct role types, which are **system**, **interface**, and **user roles**. (2) *In the agent model*, the agents are categorized into **mobile** or **stationary**. In addition, m-GAIA has the mobility model, which GAIA does not. The interaction model, the services model, and the acquaintance model are imported from GAIA into m-GAIA without change.

### Analysis Phase of m-GAIA

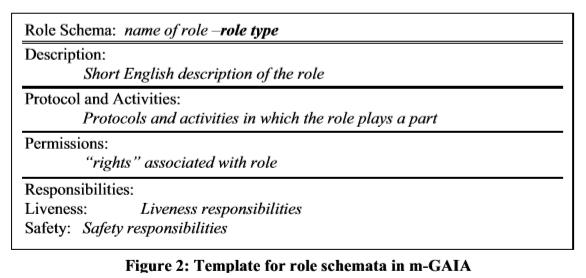
Like the GAIA methodology, the analysis phase of m-GAIA consists of the roles model and the interaction model. However, in m-GAIA, modifications have been made to roles model.

#### The Roles Model

The roles model of m-GAIA aims to identify the roles within the entire system. Each role identified is categorized into three different role *types* - **system**, **interface**, and **user roles**. The purpose of categorizing roles is to clarify each role’s responsibilities within the system.

1. A system role is defined as a role that interacts with other parts of the system and not the user.
2. An interface role is a role that interacts with the user and the other parts of the system.
3. A user role is a role that represents the human user itself.

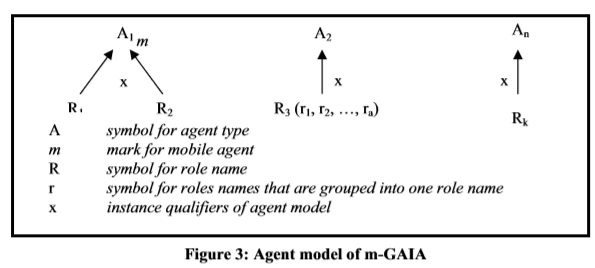
Despite the modification in the roles model, the remaining components are the same as in GAIA.



### Design Phase of m-GAIA

The design phase of m-GAIA consists of the agent model, the acquaintance model, the services model, and an additional model called the mobility model. We modified GAIA’s agent model to specify the mobility characteristic of agents. The acquaintance model and the services model are the same as those in GAIA.

#### The Agent Model

The agent model is used to identify the number of agents, the agent types, and the relationship between the roles identified (in the role model) and the agent types in the system. Unlike GAIA’s agent model, m-GAIA’s agent model classifies the agents into two different categories - **mobile** (by adding a notation of “m” sign) and **stationary**. The categorization of agent types caters for mobility characteristic of agents. Furthermore, we modify the agent model to allow similar behavior roles to be grouped into one category.

#### The Mobility Model

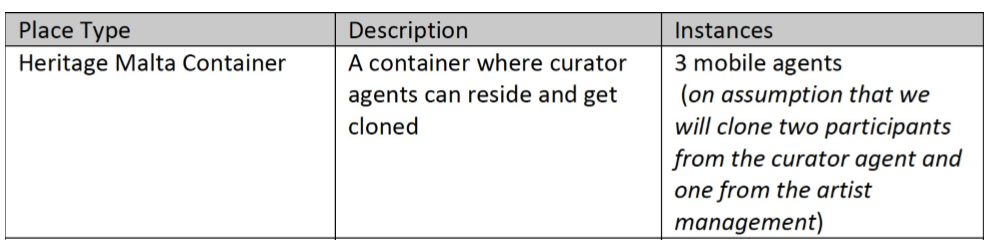
The mobility model enhances GAIA to incorporate support for modelling of mobile agents in multiagent systems. The analysis phase of m-GAIA involves identifying the roles and the interactions of each role. Unlike the analysis phase, the design phase of m-GAIA involves agents. Therefore, the mobility model is best fitted into the design phase rather than in the analysis phase, as mobility is a characteristic of agents and not roles. Furthermore, mobility is not an interaction as an agent does not need to be mobile to communicate. These considerations motivated the inclusion of the mobility model in the design phase.

The mobility model is derived from the agent model. In the agent model, the agent types are categorized into mobile and stationary. Mobile agents are able to move from one place to another place in order to perform the tasks assigned. Therefore, in order to model the mobility characteristics of mobile agents, the mobility model identifies **place types**. Place types are locations that the mobile agent can visit or reside in. The place types define the working environment of mobile agents.

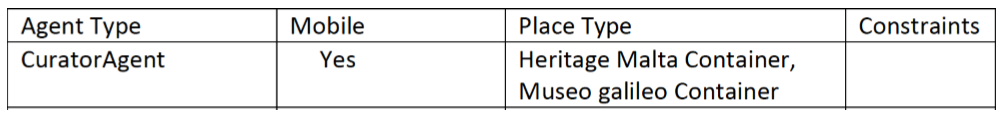
There are four steps in constructing the mobility model:

1. Identify place types.
2. Identify the relationships between agent types and place types
3. Define the cardinality between agent types and place types
4. Identify the travel path of each mobile agent.

Step 1: Place Types

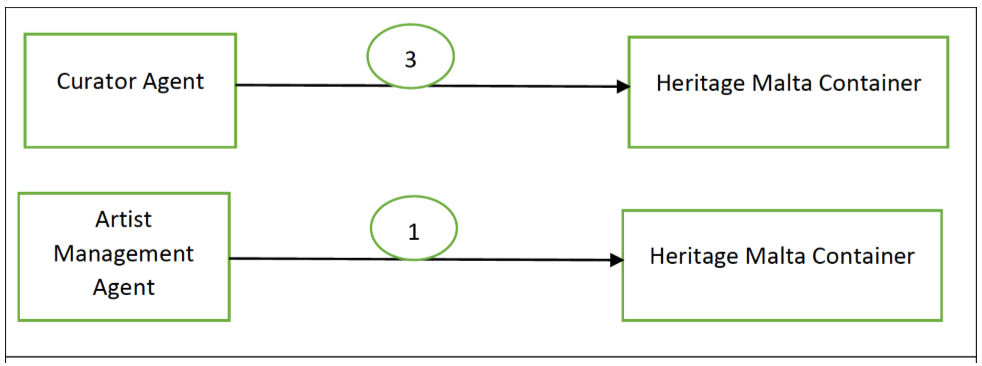


Step 2: Agents and Places Specifications

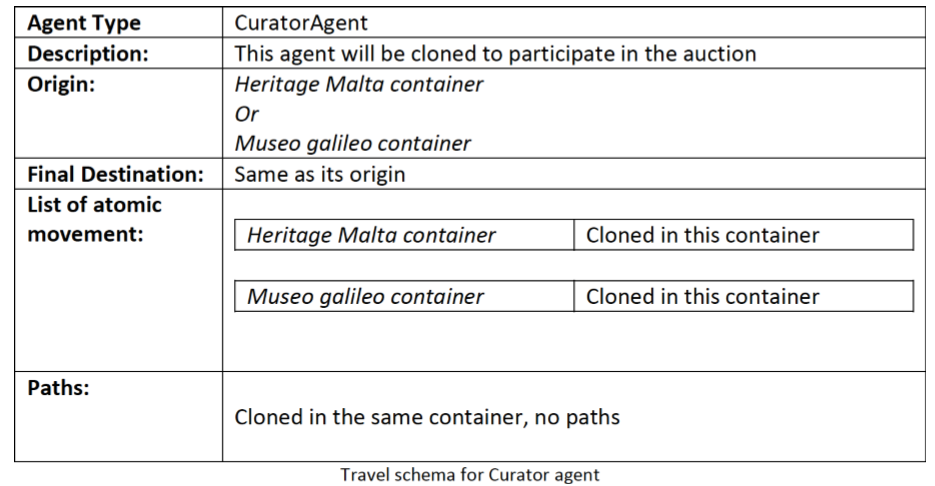
Step 2 of the mobility model is derived from step 1 and the agent model. In this step, we identify the relationship between agent types and place types. It also defines the constraints of the relationship. The agents and places specifications are derived from the place types identified in step 1 of mobility model.

Step 3: Cardinality of Agents and Places

The cardinality between agent types and place types shows how many agents of an agent type can reside in a place of a place type. The cardinality of agents and places (step 3) is based on the agents and places specifications (step 2).



Step 4: Travel Schema of Mobile Agent Types

The travel schema of each mobile agent type includes origin, final destination, list of atomic movements, and paths. The origin is the place type where the mobile agent starts the movement to accomplish the tasks assigned. The final destination is the place type where mobile agent will reside after it completed the tasks assigned. The atomic movement is the smallest granularity movement required to accomplish the tasks assigned. The paths are the list of atomic movements that the mobile agent may travel in order to accomplish the tasks assigned.