GAIA

### ****AOSE Methodologies****

* GAIA is an agent-specific methodology for agent-oriented software engineering (AOSE).
* Focuses on **organizational design** and **agent roles**, doesnt deal with specific modeling techniques or implementations.
* Best suited for closed, static systems where agent roles and relationships are predefined.

### ****Why Look at GAIA?****

* **Strengths**:
  + Conceptually grounded in multi-agent systems.
  + Treats the system as an organization with agents playing roles and interacting.
* **Limitations**:
  + Does not address requirement capture or provide implementation guidance.
  + Focuses only on design and structure.

### ****GAIA – Domain Characteristics****

* Agents are computationally significant (**powerful and capable)**
* heterogeneous

**heterogeneous** refers to the fact that the agents in a GAIA-designed system can be developed and implemented using a variety of **programming languages**, **architectures**, and **techniques**.

* Maximise some global quality, it assumes:
  + Static organizational structure (no changes in inter-agent relationships at run-time).
  + The abilities and the services of agents they provide do not change at run-time.
  + Closed systems with fewer than 100 agent types.
  + No true conflict scenarios between agents.

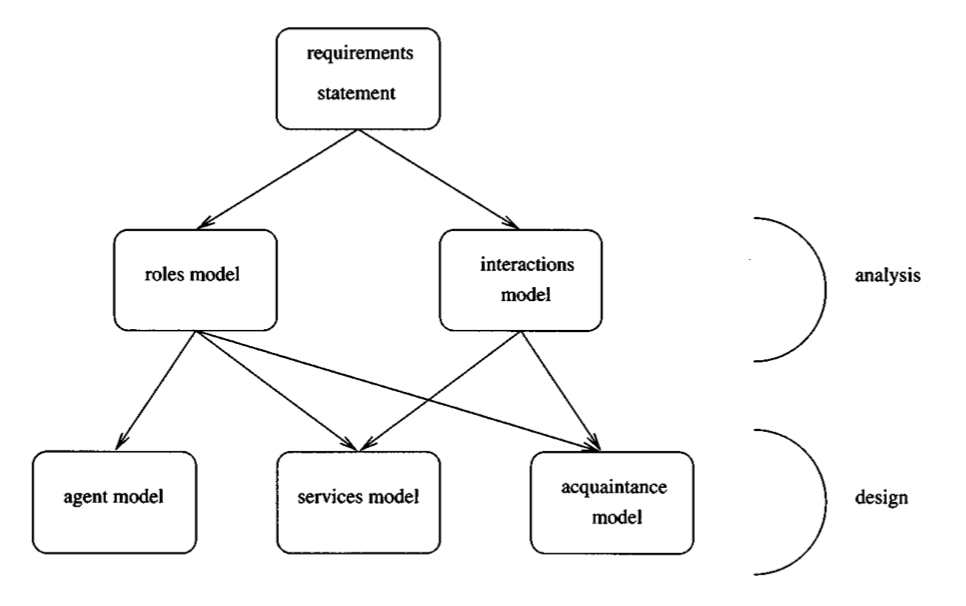
### ****GAIA & Organizational Structures****

* In the analysis phase GAIA models M.A.S. as an organization where agents play roles.

Def: A multi-agent system is comprised of a set of computational entitie**s (agents) that play roles,** and whose global system behaviour is realised through the interactions that occur between those entities.

* **Roles are** Key abstraction used for modelling the structure of an organisation.

### ****GAIA Process** (follows a organisational design)**



#### ****Analysis Phase****:

**Goal**: Identify roles, permissions, responsibilities, and interactions.

* **Outputs**:
  + **Roles Model**: Defines each role’s permissions (resource access) and responsibilities (tasks and constraints).
  + **Interactions Model**: Specifies communication protocols between roles.

A diagram of a company

Description automatically generated

This diagram illustrates the **analysis phase** of the GAIA methodology more specifically, focusing on systematically defining a multi-agent system:

1. **System**: The overarching structure and goals of the multi-agent system that we want to model. (highest-level abstraction)
2. **Role Model**: Derived from the system, roles are the fundamental units of functionality,each role has 2 attributes:

· The responsibilities of the role

· The Permissions of the role

Role Schema template helps use construct the Roles Model:

***Name****: The name of the role*

***Description****: Textual description of the role*

***Protocols and Activities****: List of the (interaction) protocols and primitive activities associated with the role*

***Permissions****: The (information) resources required by the agent and the access writes to be attributed to those resources.*

***Responsibilities****: Combines the normal behaviour of an agent playing the role and any invariants (safely conditions) that should be maintained.*

1. **Responsibilities: D**efine the functionality required by the role, split into 2 categories:
   1. **Safety properties(or responsability)**: they ensure that "nothing bad happens" (e.g., maintaining system constraints).

* they specify invariants that must be maintained during the execution of the role. *These invariants take the form of predicate expressions that define constraints*
* It is implicitly assumed that these responsibilities will apply across all states of the system execution.
* If the role is infinitely long, then the invariant will always remain true.
  1. **Liveness properties (or responsability)** : they ensure that "something good happens" (e.g., achieving goals). things that the agent must do while playing the role.
* They are represented as a liveness expression – a regular expression that takes the form: ROLENAME = expression example-> LAM = ( Notify . ReadTime . Record )^w
* The components of liveness expression are:
* Activities are fundamental actions that must be directly executable by the agent.
* Protocols define interactions with other roles and are defined in the Interaction Model.
* Common liveness expression pattern:
* Guaranteed Response type achievement goals take the form “a request is always followed by a response”.
* Infinite Repetition type achievement goals take the form “x will happen infinitely often”.

1. **Permissions**: Specify the resources and access rights required for roles to fulfill their tasks, represented as “the information or knowledge that an agent has”.

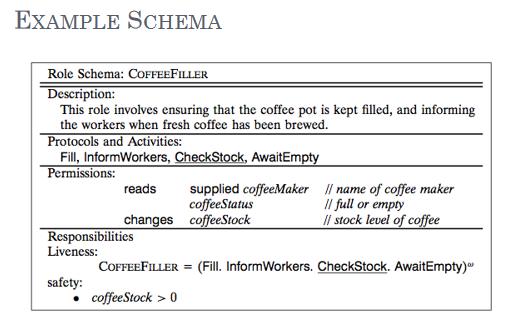
The access rights model defines how agents interact with resources. It categorizes permissions into three distinct types:

**Read Access (read)**: The role can **access the information** but cannot modify it.

**Write Access (change)**: The role can **modify existing information**.

**Create Access (generate)**: The role can **create new information**.

Example Role Schema Template



1. **Interactions Model**: Define how roles communicate and collaborate between eachother through protocols to achieve shared objectives.

* Interactions are mapped to the Protocols defined in the liveness properties of roles.
* Protocols describe structured communication patterns, such as requests, responses, or task allocations, that agents must follow.
* Interaction model should focus on the nature and purpose of the interactions rather than the specific steps
* Each interaction can be as complex or as simple as necessary
* Can involve multiple initiators or multiple responders.

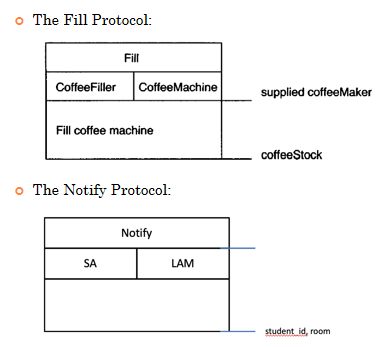
The **Interaction Model** includes a set of **protocol definitions** that specify how roles communicate and collaborate during interactions.

* purpose: brief textual description of the nature of the protocol
* 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 interaction;
* processing: brief description of any processing the protocol initiator performs during the interaction.

### ****Example Task Allocation in a Warehouse System****

* **Purpose**: To assign picking tasks to robots in an optimal manner.
* **Initiator**: The "Coordinator" agent.
* **Responder**: The "Picker" robot agents.
* **Inputs**: List of tasks, robots’ status (location, workload, battery life).
* **Outputs**: Task assignment (e.g., "Robot A assigned to Task 1").
* **Processing**: The Coordinator evaluates the inputs to determine the most efficient task allocation, then sends the assignment to the robots.

Example of Protocols



So recap of the **analysis process follow like this:**

### ****1. Identify Roles****

* output: **Unelaborated Role Model**
  + A basic list of all roles without detailed attributes, serving as the starting point.
  + Example: Roles like "Coordinator" and "Picker" in a warehouse system.

### ****2. Identify and Document Protocols****

* For each role, define protocols that describe how it interacts with other roles through communication or coordination.
* output: **Protocol Model (interactions)**
  + A structured definition of each protocol, including its purpose, initiators, responders, inputs, outputs, and processing.
  + Example: A "TaskAllocation" protocol where the "Coordinator" assigns tasks to "Picker" agents.

### ****3. Elaborate the Roles Model****

* Use the protocol model to refine and expand the roles model by specifying:
  + **Permissions, Responsibilities,Interactions**
* output: **Elaborated Role Model**
  + A detailed description of each role and its interactions, permissions, and responsibilities.

### ****4. Iterate and Refine****

* Revisit and refine steps (1)–(3) to make adjustments iteratively in protocols or roles are incorporated iteratively.

### ****Outcome****

The process delivers a clear definition of roles and their interactions, producing a robust **roles model** and **protocol model** as the foundation for system design.

#### ****Design Phase****:

* **Goal**: Translate analysis models into implementable components that is a sufficiently low level of abstraction.
* In this phase we generate:
  + **Agent Model**: It Maps roles to agent types and instances.
  + **Services Model**: Defines agent functions with inputs, outputs, and conditions.
  + **Acquaintance Model**: Specifies communication links between agents.

The Agent Model:

focuses on defining the agents that must exist and linking them to the roles they must fulfill

**1.Identifying Agent Types**:

· The model determines the **agent types** that must exist in the system.

Each **agent type represents a class or category of agents** that fulfill functions within the system.

Each agent type is a **generalized template** for creating agents with similar responsibilities and behaviors.

· These types are defined based on the **roles** identified during the analysis phase.

**2.Mapping Roles to Agent Types**:

Each agent type is defined by a set of roles, and individual **instances** (agents) of that type are responsible for fulfilling these roles.

· Instances are the **specific agents** created from the template (agent type) .

Example: If "Picker Agent" is an agent type, its instances (e.g., Robot A, Robot B) will perform the roles of retrieving items and notifying task completion.

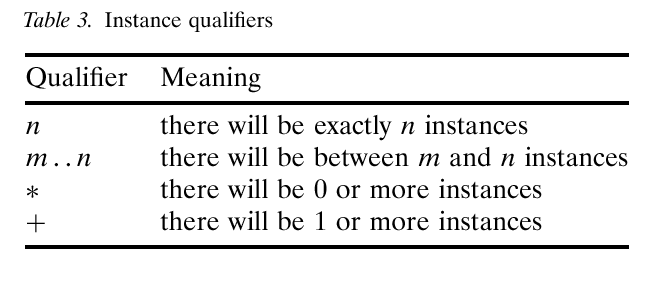
· Example: In a warehouse system:

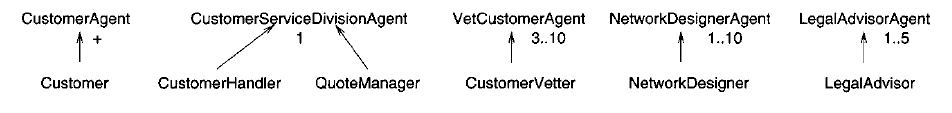
* A "Picker Agent" may fulfill the "Picker" role.
* A "Manager Agent" may combine roles like "Coordinator" and "Monitor."

**3.Representation Using Agent Type Trees**:

* Agent type trees visually represent the relationship between **agent types** and **roles**.
* Each node in the tree represents an agent type, while branches link the type to the roles it fulfills.
* This structure helps clarify which roles are grouped together within each agent type.

The model includes **instance qualifiers** to specify how many instances of each agent type will exist in the system.





The Service Model:

It provides a detailed description of each **service** ( mapped to activity) and their inputs, outputs, pre-conditions, and post-conditions.

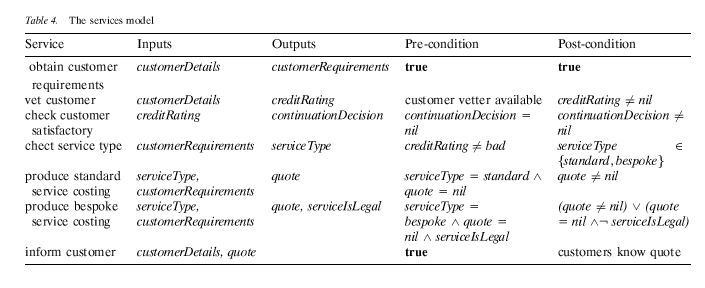
In GAIA , a service is a specific function or capability of an agent, it is what the agent actually does to carry out its assigned responsibilities within the system.

* That is, the service model is used to refine the protocols, activities and responsibilities of the agent.
* There should be at least **one service per activity**(defined in the liveness properties).
* Services **may also be associated with protocols**(defined in the liveness properties and interaction model).

For example:

The CheckStock activity 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 stock level is greater than zero.

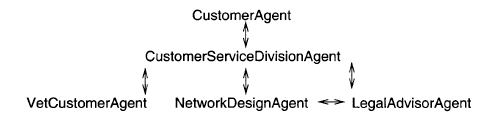
A "SubmitTask" service might be part of a protocol where a picker agent communicates task completion to the coordinator.



The Acquaintance Model:

defines communication links between agents in a multi-agent system, specifying who interacts with whom.

Example:



In the given scheme:

**CustomerServiceDivisionAgent** is the central hub, connecting with:

* 1. **CustomerAgent** for handling customer interactions.
  2. **VetCustomerAgent** for verifying customer information.
  3. **NetworkDesignAgent** for creating network solutions.
  4. **LegalAdvisorAgent** for addressing legal issues.

**NetworkDesignAgent** and **LegalAdvisorAgent** also communicate directly to ensure compliance with regulations.

The model ensures clear, bidirectional communication paths, modularity, and effective collaboration between agents, supporting scalability and transparency in system operations.

So recap of **the Designprocess follow like this:**

### ****1. Create an Agent Model****

* **Group Roles into Agent Types**: Combine logically related roles into agent types to optimize efficiency.
  + Example: A "Coordinator Agent" might combine the roles of "Task Allocator" and "Progress Monitor."
* **Form Agent Type Hierarchies**: Organize agent types to show their relationships and roles.
* **Document Instances**: Specify the number of instances required for each agent type.
  + Example: 1 "Coordinator Agent" and 10 "Picker Agents."

### ****2. Develop a Service Model****

* **Translate Activities to Services**: Convert liveness properties into actionable services.
  + Example: The activity "Record Attendance" becomes the service "LogAttendance."
* **Incorporate Protocols**: Define services to support inter-agent communication.
* **Detail Services**: Specify inputs, outputs, pre-conditions, and post-conditions for each service to ensure clarity and proper execution.

### ****3. Develop an Acquaintance Model****

* **Define Communication Pathways**: Use the Interaction and Agent Models to specify which agents communicate and how.
  + Example: A "Picker Agent" updates the "Coordinator Agent" about task progress.
* **Visualize Relationships**: Represent communication paths to show dependencies and ensure seamless collaboration.

### ****Example: Lecture Attendance Monitoring Agent****

* **Role**: Records attendance by reading student IDs and generating records.
* **Permissions**:
  + Reads student IDs and time.
  + Creates attendance records.
* **Responsibilities**:
  + **Liveness**: Ensures attendance is recorded.
  + **Safety**: Maintains accurate records.

### ****Summary****

* GAIA is ideal for modeling and designing static, closed MAS with predefined roles and relationships.
* Strengths: Structured approach focusing on roles, interactions, and organizational behavior.
* Limitations: No support for dynamic agents or runtime changes.

Prometeus?