**SENG 697**
**Agent-based Software Engineering**

*Session 2: Methodologies for agent-based analysis and design*

**Behrouz Far**
Schulich School of Engineering, University of Calgary
far@ucalgary.ca
http://www.enel.ucalgary.ca/People/far/

---

**Multi Agent system Development**

What do we need to develop an agent-based system? Agent-based development *methodology*
- How to analysis and design a multi-agent system?
  - e.g., GAIA, Tropos, Prometheus, etc.

Agent-based development *platform*
- How to implement a multi-agent system?
  - e.g., FIPA-OS, Jade, Jive, Aglets, etc.

---

**Introduction (contd.)**

- Evolution of software engineering paradigms:
  - Assembly languages
  - Procedural and structured programming
  - Object Oriented programming
  - Component-ware
  - Design patterns
  - Software architectures
  - Software Agents

---

**Why We Need This?**

- The authors have devised a methodology that has been specifically tailored to the analysis and design of agent-based systems.
- Why we need this?

---

**Why We Need This? (contd.)**

- The way Object Oriented systems are designed
- Objects and their “pre-defined” interactions are designed
- Behaviour is the result of objects interactions

---
Why We Need This? (contd.)

- The way agent-based systems are designed
- Agents, their “roles” in the society, “goals” and “possible” ways of interactions are defined.
- Behaviour is the result of agents interactions

Contents

- Session 2: Methodologies for agent-based analysis and design
  - A Methodology for Agent-Oriented Analysis and Design (GAIA)
  - Tropos
  - The MASSIVE Development Method
  - Comparison of development techniques

Abstract

- GAIA is a methodology for agent-oriented analysis and design. [detailed analysis]
- GAIA methodology is concerned with how a society of agents cooperate to realize the system level goals, and what is required of each individual agent in order to do this.
- The methodology is general. It is applicable to a wide range of multi-agent systems. It deals with both the macro-level (societal) and the micro-level (agent) aspects of systems.
- The methodology is founded on the view of a system as a computational organization consisting of various interacting roles.

Introduction

- Progress in software engineering over the past two decades has primarily been made through the development of increasingly powerful and natural abstractions with which to model and develop complex systems.
- Procedural abstraction, abstract data types, and, most recently, objects, are all examples of such abstractions.
- It is the authors’ belief that agents represent a similar advance in abstraction: they may be used by software developers to more naturally understand, model, and develop an important class of complex distributed systems.
Domain 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 maximizes some global quality measure, but which may be suboptimal from the point of view of the system components.
- Agents are heterogeneous, in that different agents may be implemented using different programming languages and techniques.
- The overall system contains a comparatively small number of agents (usually less than 100).

Abstract Concepts

- System
- Role(s)
  - Permission
  - Responsibility
  - Protocol
  - Liveness property (goal, utility)
  - Safety property (guards)

Analysis Concepts

Conceptual Framework

- The methodology 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 the methodology, the analyst moves from abstract to increasingly concrete concepts.
- Each successive move introduces greater implementation bias, and shrinks the space of possible systems that could be implemented to satisfy the original requirements statement.
- The methodology encourages a developer to think of building agent based systems as a process of organizational design.

Abstract & Concrete Concepts

- The concepts used by GAIA are split in two categories:
  - abstract: used in the analysis phase, that don’t necessary have a correspondent in the implementation of the system
  - concrete: concepts that typically have a correspondent in 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>Liveness properties</td>
<td></td>
</tr>
<tr>
<td>Safety properties</td>
<td></td>
</tr>
<tr>
<td>Protocols</td>
<td>Activities</td>
</tr>
</tbody>
</table>

1. System = Society of Agents

- The most abstract entity in the concept hierarchy is the system which is equivalent to society or organization of agents. That is, we think of an agent-based system as an artificial society or organization.
- The idea of a system as a society is useful when thinking about the next level in the concept hierarchy: roles.

2. Role

- It may seem strange to think of a computer system as being defined by a set of roles, but the idea is quite natural when adopting an organizational view of the world.
- Consider a human organization such as a typical company. The company has roles such as president, vice president, etc.
- The roles will be instantiated with actual individuals: i.e., individuals take on the role of president, vice president, etc.
- The instantiation is not necessarily static. E.g., throughout the company lifetime, many individuals may take on the role of company president.
- A role is defined by three attributes: responsibilities, permissions, and protocols.
3. Responsibilities (What?)
- Responsibilities determine functionality and are the key attribute associated with a role.
- **Example:** a 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*.

4. Liveness (Goal or Utility)
- *Liveness properties* describe what must be achieved by a role under certain environmental conditions.
- **Example:** increase profit for “seller” role. Minimize the price while getting the best quality for “buyer” role.

5. Safety (Guards)
- *Safety properties* impose a number of constraints on the behavioural states of a role (i.e., that an acceptable state of affairs is maintained across all states of execution).
- **Example:** Ensure the liabilities of the company are always less than assets.

6. Permissions = Rights
- In order to realise responsibilities, a role is usually associated with 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.
- In typical systems, permissions tend to be information resources.
- **Examples:**
  - Ability to retrieve a particular information; modify or delete another piece of information.
  - Ability to generate information.

7. Protocols (How?)
- A role is also identified with a number of protocols, which define the way that a role can interact with the other roles in the system.
- **Example:**
  - A *seller* role might have the protocols *Dutch auction* and *English auction* associated with it.
  - A *hiring* role of the president of a company is to conform to the hiring protocol (policies?).
How to Find Roles?

- Use-Case Realization
- Supplementary Specifications
- Glossary
- Use-Case Model
- Software Architecture Document

Views:
- Conceptual view
- Module view
- Code view
- Execution view

Supplementary Specifications

Glossary

Use-Case Model

Software Architecture Document

Roles

Actors

Use cases

Views:

Case of Agile Development

- Main source: User stories

Example:

<table>
<thead>
<tr>
<th>Customer</th>
<th>BSC Clerk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>After inspecting/completing a request, can mark the status as held/complete. If “hold”, can enter the reason. If “complete”, must have all documents attached.</td>
</tr>
<tr>
<td>Estimate</td>
<td>4.0 hours</td>
</tr>
<tr>
<td>Acceptance Tests</td>
<td>1. Mark as complete with required documents not included. (Full)</td>
</tr>
</tbody>
</table>

Analysis & Design Models

- Requirements Statement
- Roles Model
- Interactions Model
- Agent Model
- Services Model
- Acquaintances Model

Analysis

Design

aka. Detailed analysis

1) Analysis Phase

- The objective of the analysis phase is to develop an understanding of the system and its structure as a multi-agent organization. This understanding is captured in the system organization.
- Organization is viewed 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.
- To define an organization, it suffices to define the roles in the organization, how these roles relate to one another, and how a role can interact with other roles.
- The analysis phase is comprised of two models:
  - Role model
  - Interactions model

Analysis: 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 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. 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.

Analysis: 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 be spent while carrying out the role.
- In this method, in order to carry out a role, an agent will typically be able to access certain information (or other resources).
- Some roles might generate information; others may need to access a piece of information but not modify it, and others may need to modify the information.
The functionality of a role is defined by its responsibilities. These responsibilities can be divided into two categories: liveness responsibilities and safety conditions.

**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.

**Example:** The guaranteed response type of achievement goal has the form “request is always followed by a response”.

Liveness expressions define the potential execution trajectories through the various activities and interactions (i.e., over the protocols) associated with the role.

---

**Analysis: Responsibilities**

- In many cases, it is insufficient simply to specify the liveness responsibilities of a role. This is because an agent, carrying out a role, will be required to maintain certain invariants while executing.

**Example:** We might require that a particular agent taking part in an electronic commerce application never spends more money than it has been allocated.

These invariants are called safety conditions, because they usually relate to the absence of some undesirable condition arising.

Safety requirements are specified by means of a list of predicates. These predicates are typically expressed over the variables listed in a role’s permissions attribute.

---

**GAIA: Analysis Process**

1. Identify the roles in the system.
   - **Output:** A prototypical roles model, a list of the key roles that occur in the system, each with an informal, unelaborated.

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.
   - **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, and the protocols in which they take part.

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

---

**Example System**

- **Multi-Agent Dispatch System (MADS)** provides Road-side assistance system for a fleet of vehicles

---

**Example: MADS System**

- **Role: Geocode**
  - **Description:** wraps the MapPoint geocoding Web Services and it queries MapPoint for geocodes.

- **Role: RenderMap**
  - **Description:** It wraps the MapPoint map rendering Web Services and it queries MapPoint for maps.

- **Role: Route**
  - **Description:** It wraps the MapPoint routing Web Services and it queries MapPoint for routes.

- **Role: PersonalAssistant**
  - **Description:** It acts on behalf of a system user. It gets the geocode information, maps and routes, and it calculates distances between the breakdown and available service providers.

---

**Example (cont’d)**

**Permissions for RenderMap:**

- **reads**
  - **MapPointWebServices**
  - **newMap** // true or false

- **changes**
  - **notifyUser** // true or false
Example (cont’d)

Liveness Property:
- When the newMap is true, get a new map.
- When the new map is ready, notify the user.

```
RENDERMAP = (RenderMap)
RENDERMAP = RequestMap.QueryMapPoint.RespondMap
```

Example: RenderMap

```
Role Schema
<table>
<thead>
<tr>
<th>Role Schema</th>
<th>RenderMap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>It wraps the MapPoint map rendering Web Services and it queries MapPoint for maps.</td>
</tr>
<tr>
<td>Protocols and Activities</td>
<td>QueryMapPoint, RequestMap, RespondMap</td>
</tr>
<tr>
<td>Permissions</td>
<td>reads MapPoint Web Services</td>
</tr>
<tr>
<td>Responsibilities</td>
<td>Liveness: RENDERMAP = (RenderMap), Safety: a successful connection with MapPoint Web Services</td>
</tr>
</tbody>
</table>
```

Analysis: Interaction Model

- There are dependencies and relationships between the various roles in a multi-agent organisation. Interactions need to be captured and represented in the analysis phase.
- This model consists of a set of protocol definitions (mechanisms), one for each type of inter-role interaction.
- A protocol can be viewed as a pattern of interaction that has been formally defined and abstracted away from any particular sequence of execution steps.
- A single protocol definition will give rise to a number of message interchanges in the run time system.
- Example: English auction protocol. This involves multiple roles (sellers and bidders) and many potential patterns of interchange (price announcements and corresponding bids). However at the analysis stage, such precise instantiation details are not necessary.
**Example: Interaction Model**

<table>
<thead>
<tr>
<th>Protocol</th>
<th>RequestGeocode</th>
<th>RequestMap</th>
<th>RequestRoute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose/parameters</td>
<td>Request to geocode a location. The response includes the Latitude and Longitude of the location</td>
<td>Request a map containing a list of locations. The response includes the URL to the map that displays the locations</td>
<td>Request a route between two locations. The response includes the driving direction between the locations</td>
</tr>
<tr>
<td>Initiator(s)</td>
<td>PersonalAssistant</td>
<td>PersonalAssistant</td>
<td>PersonalAssistant</td>
</tr>
<tr>
<td>Receiver(s)</td>
<td>Geocode</td>
<td>RenderMap</td>
<td>Route</td>
</tr>
<tr>
<td>Responding Protocol</td>
<td>RespondGeocode</td>
<td>RespondMap</td>
<td>RespondRoute</td>
</tr>
</tbody>
</table>

**2) Design Phase**

- The aim of the design phase is to transform the analysis models into a lower level of abstraction that traditional design techniques (e.g. object-oriented techniques) may be applied.
- How an agent realizes its services and how it is implemented is beyond the scope of the methodology, and will depend on the particular application domain.

**Analysis & Design Models**

- **Requirements Statement**: Describes the purpose of the system and the expected outcomes.
- **Agent Model**: Defines the agent types that will be used in the system.
- **Services Model**: Identifies the services that will be provided by the agents.
- **Acquaintances Model**: Documents the relationships between agents.
- **Design Models**: Composed of agent, service, and acquaintance models.
- **Detailed Analysis**

**Design: Agent Model**

- The purpose of the **agent model** is to document the various agent types that will be used in the system, and the agent instances that will realise these agent types at run-time.
- An **agent type** is composed of 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. A designer can choose to package a number of closely related roles in the same agent type for the purposes of convenience.
- The agent model is defined using a simple tree, in which root nodes correspond to agent types and leaf nodes correspond to roles, (as defined in the roles model).

**Agent Model**

```
AgentType1
    * 
    / 
Role1 Role2
    * 
    / 
Role3 
```

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>there will be exactly n instances</td>
</tr>
<tr>
<td>m .. n</td>
<td>there will be between m and n instances</td>
</tr>
<tr>
<td>*</td>
<td>there will be 0 or more instances</td>
</tr>
<tr>
<td>+</td>
<td>there will be 1 or more instances</td>
</tr>
</tbody>
</table>
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 from the protocols model.

Pre-conditions and post-conditions represent constraints on services. These are derived from the safety properties of a role. 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 and responsibilities associated with a role, and in particular, from the liveness definition of a role.

Acquaintance models define the communication links that exist between agent types. They do not define what messages are sent or when messages are sent. They indicate that communication pathways exist.

The purpose of acquaintance model is to identify any potential communication bottlenecks, which may cause problems at run-time. Helpful when testing the system.

An agent acquaintance model is 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: A \rightarrow B indicates that A will send messages to B, but not necessarily that B will send messages to A.
Design Process

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 protocols and safety and liveness properties of roles.
3. Develop an acquaintance model from the interaction model and agent model.

Extensions (Future Works)

- Dealing with truly open systems, in which agents may not share common goals (liveness property). This class of systems represents the most important application area for multi-agent systems, and it is therefore essential that the methodology be able to deal with it.
- Notion of an organizational structure: at the moment, such structures are only implicitly defined within the methodology via the role, interaction and acquaintance models. However, direct, explicit representations of such structures will be of value. Representing such structures may be the only way of adequately capturing and understanding the organization’s communication and control structures.

Critics

Evaluate the methodology along with the followings:
- Roles vs. tasks: yes
- Emergent behaviour: n/a
- Knowledge cycle: no
- Interactions vs. intra-actions: yes
- Symbol level communication: no (may be)
- Knowledge completeness: no
- Indeterminism: Decision making: no

Evaluation /1

- Concepts: abstract and concrete concepts are clearly defined. (Good)
- Modeling constructs: uses a clear and consistent notation with a clear and defined syntax and semantics. None of the abstracts and concrete concepts or the models are in conflict and they can be interpreted unambiguously. (Good)
- Scalability: Roles are not hierarchical. The role model provides strictly one level of abstraction for the developers to conceptualize the system. The methodology does not facilitate iterative refinement of the system through different levels of abstraction. As a result, GAIA does not scale up well to handle complex systems. (Poor)

Evaluation /2

- GAIA does not have the ability to track dependencies between different models and the underlying code used to implement them.
- GAIA does not directly deal with particular modeling techniques or implementation details.
- GAIA models do not specify what tasks are done in parallel, so it has a weak support for concurrency.
- GAIA offers no mechanisms to model the dynamic reasoning, extension and modification of the social aspects at runtime.
- GAIA assumes complete specification of requirements and does not address the requirement gathering phase.

Evaluation /3

- Limited process life-cycle coverage: 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, but ignoring implementation, testing/debugging, deployment and maintenance details.
- Limited process management: GAIA does not address any management and/or cost estimation aspects involved in software development.
Evaluation /4

- **Formal semantics:** GAIA does not have a formal semantics. A successful methodology is one that is not only of pragmatic value, but one that also has a well-defined, unambiguous formal semantics. While the typical developer need never even be aware of the existence of such a semantics, it is nevertheless essential to have a precise understanding of what the concepts and terms in a methodology mean.

Conclusions

- GAIA is a “popular” methodology for analysis and design of agent-based systems.
- The key concepts in this methodology are **roles**, which are associated with **responsibilities, permissions**, and **protocols**. Roles can interact with one another in certain institutionalised ways, which are defined in the protocols of the respective roles.

The Tropos Development Methodology

- **Tropos** is derived from the Greek *trope* which means “easily changeable” or “easily adaptable.”
- The Tropos Software Methodology was first presented in 2001 at the at 5th IEEE International Symposium on Requirements Engineering where the authors proposed a new specification language, called Formal Tropos, that offers the primitive concepts of early requirements frameworks.

Fundamentals

- Tropos has two main features:
  - Agents, goals, plans, and other knowledge level concepts are fundamental primitives in their notation and used informally during the complete software development process.
  - Requirements analysis and specification plays an important role when the system that has to be developed is analyzed with respect to its intended environment.
Comparison

- Tropos is used during the whole software development process.

<table>
<thead>
<tr>
<th></th>
<th>Early Requirements</th>
<th>Late Requirements</th>
<th>Architectural Design</th>
<th>Detailed Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tropos</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KAOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAIA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AML andMatrixMode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conceptual Entities /1

- Symbolic representation of the main concepts of Tropos

- Actor
  - Goal
  - Softgoal
  - Resource
  - Plan

Goal dependency

Depends on Dependee

1. Actor & Dependency Model

- Several models can be built using Tropos conceptual entities

Tropos: Methodology

- The phases during software development that are covered by Tropos are the following:
  - **Early Requirements**: This phase includes identification of relevant stakeholders (actors) with respective objectives (goals).
  - **Late Requirements**: The target system is introduced as another actor and is related to stakeholder actors in terms of actor dependencies (these indicate the obligations of the system towards its environment).
  - **Architectural Design**: Additional system actors are introduced and assigned to goals or subtasks of system goals or system tasks.
  - **Detailed Design**: All system actors are described in detail (including specifications of communication and coordination); output is a Tropos specification.
  - **Implementation**: The Tropos specification is transformed to a skeleton for implementation (in detail, this is done by mapping from Tropos constructs to those constructs of an agent programming platform; later code is added to the skeleton).
Sample Actor Diagram

Dependency between two actors: one actor depends on the other to attain some goal, execute some plan or deliver a resource. First actor is called depender, second one is dependee. The object around which the dependency centers is called dependum.

2. Goal and Plan Models

A designer can analyze goals and plans from an actor’s perspective using three techniques:
- **Means-ends analysis.** Refining a goal into subgoals to identify plans, resources, and softgoals (which provide means for achieving the goal at the end).
- **AND/OR (de)composition.** Allows a combination of AND and OR (de)compositions of a root goal into subgoals (creates a goal structure).
- **Contribution analysis.** To point out goals contributing negatively or positively in reaching the root goal being analyzed. It is a special case of means-end analysis where means are always goals.

Goal and Plan Models

- Samples of Means-End Analysis, OR-Decomposition, And-Decomposition, Contribution, and Actor perspective.

Tropos: Process

* Tropos is a generic design process which focuses on a goal analysis for different actors. The process is a non-deterministic concurrent algorithm (including a completeness criterion). During the analysis process the design flows from external to system actors through a goal analysis and delegation.
- The process starts with actors with a list of related goals (and softgoals) and the root goals are analyzed from the actor’s perspective. Generated subgoals are delegated to other actors or the actor itself when it deals with the goals itself. This analysis is carried out concurrently with respect to each root goal. The process may require the introduction of new actors which are delegated to goals and/or tasks. The process ends when all goals have been dealt with to the satisfaction of actors.

Formal Description of Process

```
global actorList, goalList, agenda, dependencyList, capabilityList, goalGraph;
procedure rootGoalAnalysis(actorList, goalList, goalGraph)
begin
rootGoalList = nil;
for actor in actorList do
    for rootGoal in goalList(actor) do
        rootGoalList = add(rootGoal, rootGoalList);  
        rootGoal.actor = actor;
    end;
end;
concurrent for rootGoal in rootGoalList do
    goalAnalysis(rootGoal, actorList);
end concurrent for;
if not satisfied(rootGoalList, goalGraph) then fail;
end procedure
```
Formal Process: Explanation

<table>
<thead>
<tr>
<th>actorList</th>
<th>Finite set of actors.</th>
</tr>
</thead>
<tbody>
<tr>
<td>goalList</td>
<td>List of goals for actor (in goalList(actor)).</td>
</tr>
<tr>
<td>agenda</td>
<td>List of goals actor has undertaken to achieve personally (initially empty).</td>
</tr>
<tr>
<td>dependencyList</td>
<td>List of dependencies between actors.</td>
</tr>
<tr>
<td>capabilityList</td>
<td>Includes &lt;goal,plan&gt; pairs indicating means by which the actor can achieve goals.</td>
</tr>
<tr>
<td>goalGraph</td>
<td>Stores representation of goal graph that has been generated so far by the design process (initially contains all root goals of all initial actors with no links among them).</td>
</tr>
<tr>
<td>rootGoalAnalyse</td>
<td>Conducts concurrent goal analysis for every root goal.</td>
</tr>
<tr>
<td>satisfy</td>
<td>Checks whether all root goals in goalGraph are satisfied.</td>
</tr>
<tr>
<td>goalAnalyse</td>
<td>Conducts concurrent goal analysis for every sub goal of a given root goal.</td>
</tr>
</tbody>
</table>

Tropos: Modeling Language

- **Meta-metamodel level.** The meta-metamodel level is the basis for the extensions of the metamodel. It contains the language primitives that allow the inclusions of constructs (basic language, structural elements, e.g. attributes, entities).
- **Metamodel level.** The metamodel level provides constructs for modeling knowledge level entities and concepts (knowledge level notations, e.g. actor, goal, dependency).
- **Domain model level.** The domain model level provides a representation of entities and concepts of specific application domains which are built as instances of the metamodel constructs (application domain entities).
- **Instance Model level.** The instance model level contains instances of the domain model.

Tropos: Modeling Language

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta-Metamodel</td>
<td>Specifies language structural elements</td>
<td>Attribute, Entity</td>
</tr>
<tr>
<td>Metamodel</td>
<td>An instance of the meta-metamodel</td>
<td>Actor, Goal, Plan</td>
</tr>
<tr>
<td>Domain</td>
<td>Defines knowledge level notions</td>
<td></td>
</tr>
<tr>
<td>Domain</td>
<td>An instance of the metamodel</td>
<td></td>
</tr>
<tr>
<td>Domain</td>
<td>Models application domain entities</td>
<td></td>
</tr>
<tr>
<td>Instance</td>
<td>Instantiates domain model elements</td>
<td></td>
</tr>
</tbody>
</table>

Conclusions

- Tropos is one of the “better” methodologies in terms of blending agent-based technology and contemporary software engineering practice.

Design Level

- At the design level several design patterns, such as **Pair** pattern and **Mediation** patterns can be used to implement the system.
  - The **Pair** patterns – such as booking, call-for-proposal, subscription, or bidding – describe direct interactions between negotiating agents.
  - The **Mediation** patterns – such as monitor, broker, matchmaker, mediator, embassy, or wrapper – feature intermediary agents that help other agents to reach an agreement on an exchange of services.
  - Etc.

The MASSIVE Development Method for Multiagent Systems

Jurgen Lind
Abstract

- MASSIVE is a pragmatic process model for the development of multiagent systems based on the combination of standard software engineering techniques.
- The development method features a **product model** to describe the target system, a **process model** to construct the product model and an **institutional framework** that supports learning and reuse over project boundaries.
- The approach has evolved over several years and it has been (successfully?) applied and refined in various industrial research projects.

Introduction

- Late 90’s: The strong need for agents and agent systems development methods in an industrial context is exemplified by the major European telecommunication companies that have launched a joint research project to foster the definition of a development method for agent applications.

Software Development Methods

- A software development method encompasses a notation, whose purpose is to provide a common means of **what** to be developed and specifying **how** (and **when**) certain artifacts should be produced.
- Recently, several development methods for multiagent system have been proposed. However, most of these methods feature a **biased** and **technology-driven**, **model-oriented** and **sequential approach** towards the development of multiagent applications that is not always adequate for this class of systems.

Requirements-driven vs. Technology-driven

- Most of the related development methods assume that the technology, i.e. a multiagent approach, for the target system has already been set and that the system must be designed and implemented according to this technology.
- In a requirements-driven development approach, the requirements of the target system determine the technology that is to be used for the system design and implementation and not vice-versa.
- Thus, a requirements-driven approach starts earlier in the life cycle of the target system when the fundamental characteristics of the problem under consideration are analyzed.

View-oriented vs. Model-oriented

- The model-oriented approaches develop several distinct models of the target system that must be integrated in order to construct the final system. This integration process is not trivial as the links between the models are usually neither explicitly defined nor obvious.
- A view-oriented method deals with the system as a whole and uses different perspectives on the entire system as fundamental abstraction. The system is not decomposed but rather viewed from different angles with different foci on particular aspects.
- The advantage of this approach is that it avoids the integration process because the model is always consistent from any view as changes in one view are always directly propagated to other views.

Iterative vs. Sequential

- A sequential process model has the assumption, that everything will work out as desired and that all documents are complete, correct and consistent once they are created. However, this assumption is unlikely that it holds in the case of multiagent systems.
- A process model that is to be used for multiagent applications should respect the fact that the system requirements and design documents are incomplete.
- It is a good idea to iterate the processes that were applied to construct the documents and to evaluate the results of each iteration according the quality measures defined early in the development process.
- This approach reduces the risk of errors because of the repeated quality evaluations.
Balanced vs. Biased

- Traditional SE methods are often biased towards the model aspects of the problem in question and neglect the code part of the resulting system.
- Novel trends in the SE community and especially the Object-Oriented, feature a point of view that treats the model and the code that implements the model as equally important parts of the whole.
- In MASSIVE this point-of-view is adopted and a development method that clearly relates the parts of the model and the matching parts of the code is presented.

The Development Framework

- The MASSIVE framework provides a notation in the form of one or more
  - Product models that are used to describe the design and/or the implementation of a particular system.
  - Process models that guide the designer in constructing the product models according to the user requirements.
  - Institutional framework that tells the project management how the design process must be configured to yield the best possible result.

MASSIVE Views

- A view is a perspective of the model that is meaningful to specific stakeholders.
- A view allows stakeholders to see the system from a perspective most meaningful to them.
- The MASSIVE method is based on seven views where each view represents a set of conceptually linked features.

Comparison: UML Views

View System Requirements

- A view system for the development of multiagent applications should comply to the following requirements:
  - Separation: cohesion and coupling; decomposability
  - Coverage: integrity
  - Flexibility
  - Size: Granularity
  - Naming

1. Task View

- In the Task view, the functional aspects of the target system are analyzed and a task hierarchy is generated that is then used to determine the basic problem solving capabilities of the entities in the final system.
- Furthermore, the non-functional requirements (e.g., performance, compatibility, security, reliability, usability) of the target system are defined and quantified as far as possible.
- This view does not assume that a multiagent approach is used for the final system.
- Example: In a compiler application, the functional requirement is that the system translates a program specified in a high-level language to a particular assembly language. The quality of the resulting code or the maximal tolerable time for the compilation are non-functional requirements and the basic problem solving capabilities are lexical analysis or code generation.
2. Environment View

- In environment view, the environment of the target system is analyzed form the developers perspective and from the systems perspective.
- These two points-of-view usually differ greatly as the developer usually has global knowledge whereas the system has only local knowledge.
- Example: In a multiagent system, an individual agent has access to only parts of the environment and resources and there are ongoing activities that cannot be perceived by the agent.

3. Role View

- Role view determines the functional aggregation of the basic problem solving capabilities according to the physical constraints of the target system.
- A role is an abstraction that links the domain dependent part of the application with the agent technology that is used to solve the problem under consideration.
- An agent consists of:
  - One or more role descriptions and
  - An architecture that is capable of executing these role models

4. Interaction View

- Interaction view is the fundamental concept within a system that is comprised of multiple independent entities that must coordinate themselves in order to achieve their individual as well as their joint goals.
- In this view, interaction within the target system is seen as a generalized form of conflict resolution that is not limited to a particular form such as communication.
- Several generic forms of interaction that can be instantiated in a wide variety of contexts are discussed and the developer is encouraged to analyze the target problem with respect to the applicability of these generic forms before designing new forms.

5. Society View

- A society is a structured collection of entities that pursue a common goal.
- The goal of this view is to classify the society that either pre-exists within the organizational context of the system or that is desirable from the point of view of the system developer.
- A society model is developed that is consistent with the roles within the society and that achieves the defined goals.
- The final structure of the agent society (flat or clustered) thus depends on the precedence of the quality measures (quality of the solution vs. efficiency) that is taken by the system designer.

6. Architectural View

- The architectural view is a projection of the target system onto the fundamental structural attributes with respect to the system design. The major aspects that are dealt with in this view are the system architecture as a whole and the agent architecture.

  System architecture is described according to various considerations and includes things such as the agent management or database integration.

  Agent architecture is characterized according to the requirements of the problem to be solved.

- System developers should at first try to select one of the numerous existing architectures (platforms) before trying to develop a new architecture from scratch.

7. System View

- System view deals with aspects that affect several of the other views at the same time or even the system as a whole.
- System view handles: user interface that controls the interaction between the system and the user(s) on a task specific as well as the task independent level.
- Task specific aspects are usually input specification and output presentation.
- Task independent aspects deal with the visualization of the system activities in order to enable the user to follow the ongoing computations and interactions.
- Other aspects that are treated in this view are: system-wide error-handling strategy; performance engineering; and system deployment once it has been developed.
Iterative View Engg /1

- Iterative view engineering is a product centered SE process model that combines Round-trip Engineering and Iterative Enhancement.
- Iterative Enhancement is a process model that is similar to the Waterfall Model except that the Waterfall Model needs a complete and stable requirements definition.
- Iterative Enhancement model operates in several cycles on a partitioned and incomplete system model. The incomplete model is iteratively enhanced in each cycle.

Iterative View Engg /2

- See the Iterative Enhancement as the macro process that encapsulates several micro processes that are used to construct individual parts of the model.
- In the example figure, product model consists of three parts and each part is iteratively constructed with another process model. The vertical boxes represent a snapshot of the entire model at particular points in time t1 and t2 where the parts of the model can be in different process stages.

Iterative View Engg /3

The Iterative view engineering process model is the following:

1. Initially, the model and the implementation are empty. In a first cycle of iterative enhancement (1), the designer specifies the first version of the model probably by using different micro-models for each view.
2. In step (2), parts of the initial model are implemented and if an error occurs during the construction phase, the model has to be refined (3) until it can be implemented.
3. Next, the initial implementation is tested. During the test phase, it often turns out that the model specification was incomplete and the implementation must be changed or enhanced (4) in order to meet the intentions of the initial model.

Iterative View Engg /4

Experience Factory /1

- The Experience Factory (BASILI, 1994) is a conceptual framework that supports systematic learning within an organization.
- The basic idea underlying the Experience Factory is closely related to case-based reasoning and operates on similarities of software development processes.
- The major advantage of the Experience Factory is that it provides a single interface to the knowledge that exists within an organization and a central authority that collects and updates this knowledge.
The core of the MASSIVE method is the view system.

The main idea of the process model is an iterative view engineering approach that is itself based on iterative enhancement and round-trip engineering.

These micro-models as well as the overall process model are in no way fixed for the entire lifetime of the project model, they are also subject to changes and refinements during the course of time.

In order to preserve these adaptations and to make them accessible to others, the process model and the product model are both embedded into a larger organizational structure called the Experience Factory.

The Experience Factory provides the formal framework for a permanent learning process that takes place over project boundaries and that eventually models the multiagent experience of organization in terms of specific product and process models for various domains.

The author presented a pragmatic software engineering development (process) model that is supposed to be specialized for the development of multiagent systems.

The model features a view oriented product model, an iterative process model and an institutional framework that supports organizational learning and reuse over project boundaries.
Critics
Evaluate the methodology along with the followings:
- Roles vs. tasks yes/no
- Emergent behaviour no
- Knowledge cycle no
- Interactions vs. intra-actions yes
- Symbol level communication no
- Knowledge completeness yes
- Indeterminism: Decision making no

A Statistical Approach for Comparing Agent Based Methodologies

Motivation /1
- Increased interest in MAS → large number of AOSE methodologies
- AOSE methodologies all utilize agent concepts and offer techniques
  - Suitable for different stages of software development process
  - Tailored to different levels of abstraction

Motivation /2
- From literature, we enumerated more than 30 AOSE methodologies
- Some methodologies do not have adequate capabilities to model the properties of all types of agents
- Some methodologies are developed for special purposes
- Some methodologies do not have sufficient documentation or presented in languages other than English

Research Question
- What is the best AOSE methodology for a given set of requirements?

The ambiguity and lack of standardization associated with the existing AOSE methodologies makes it difficult to take decisions to select the most appropriate methodology that best fits a particular agent-oriented application

Related Works /1
- In the literature, a limited number of studies have been carried out to challenge the problem of evaluating and comparing AOSE methodologies through a number of frameworks
- However, they may not necessarily follow a reliable approach assuring effective evaluation and accurate decisions

See references for a few related works
**Related Works**

- Most of the evaluation studies pay attention to highlight the best features of the presented methodologies.
- Most of the evaluation studies that we found have been conducted by the developers of methodologies. This may create a bias that makes developers unable to recognize drawbacks with their own developed methodologies.

**Research Objective**

- The main objective of this research is to devise a technique for evaluating AOSE methodologies and come up with a set of facts that support the process of selecting the most appropriate methodology to accommodate the anticipated criteria of new agent-based applications.

**Research Questions**

1. What are the major criteria upon which we can describe and evaluate an AOSE methodology?
2. What are the different features or attributes that specify each of these criteria?
3. How can we quantify and assess the evaluation criteria of AOSE methodologies, as well as their descended attributes?
4. What are the software metrics that can be used to evaluate an AOSE methodology?
5. How can we determine whether significant differences exist between the evaluated AOSE methodologies?
6. How can we rank the evaluated AOSE methodologies in order to select the most appropriate methodology for a given MAS application?

**Methodology**

- Our methodological approach can be divided into the following 4 major steps:

**Step 1: Surveying Available Methodologies**

- We started the research by conducting a comparative survey to review most of the available AOSE methodologies that are presented in literature. As a result, we came up with 31 methodologies that have reasonable documentation to study.

**Step 2: Selecting Qualified Methodologies**

- We defined a qualified methodology as a methodology that meets the following prerequisites:
  - The methodology has reasonable documentation
  - The methodology is fairly known to the agent community
  - The methodology is not a special-purpose one rather, it has a reasonable domain of applicability
Methodology /3

By reviewing the available 31 methodologies against these assumptions, the following 9 methodologies have been selected as the most qualified ones to carry on through the evaluation process:

- **Gaia** [Wooldridge 2000],
- **MaSE** [DeLoach 2002],
- **Tropos** [Giunchiglia 2002],
- **Agent-SE** [Far 2002],
- **MASSIVE** [Lind 2002],
- **Prometheus** [Padgham 2002, Dam, 2003],
- **MESSAGE** [Evans 2001],
- **MAS-CommonKADS** [Iglesias 1997], and
- **PASSI** [Cossentino 2005]

Methodology /4

**Step 3: Applying MWAF**

- We proposed the Multidimensional Weighted-Attributes Framework (MWAF), as a general-purpose framework that utilizes statistical methods for evaluating software in general, and AOSE methodologies in particular

**Step 4: Statistical Analysis**

**Methodology: MWAF /1**

Components of MWAF:
1. Dimensions
2. Attributes
3. Parameters

**Methodology: MWAF /2**

- The main idea of MWAF is to define the most common and important criteria (we referred to as dimensions) of the software being evaluated,
- Identifying the attributes that describe each of the dimensions, and then
- Evaluating each dimension through its attributes against all the potential AOSE methodologies that are given for evaluation.

**Methodology: MWAF /3**

- We identified 8 dimensions that best describe an AOSE methodology

We can break down these dimensions into the hierarchical level of their descended attributes, as follows:

**Rates & Weights**

- “Rate” reflects the extent the attribute is supported in the methodology. (how the attribute is effectively supported?)
- “Weight” reflects the extent that the attribute is important to agent-based software design and development. (how important is to MAS development?)
Methodology: MWAF /4
- Dimension 1: Agency

Methodology: MWAF /5
- Dimension 2: Modeling

Methodology: MWAF /6
- Dimension 3: Communication

Methodology: MWAF /7
- Dimension 4: Process

Methodology: MWAF /8
- Dimension 5: Upgrade

Methodology: MWAF /9
- Dimension 6: Application
Methodology: MWAF /10

- Dimension 7: Supporting Properties

Methodology: MWAF /11

- Dimension 8: User Perception

Methodology: MWAF /12

|   |    |    |    |    |    |    |    |    |
|---|----|----|----|----|----|----|----|
|   | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 |
|   |    |    |    |    |    |    |    |    |
|   | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  |
|   |    |    |    |    |    |    |    |    |
|   |    |    |    |    |    |    |    |    |
|   |    |    |    |    |    |    |    |    |
|   |    |    |    |    |    |    |    |    |
|   |    |    |    |    |    |    |    |    |
|   |    |    |    |    |    |    |    |    |
|   |    |    |    |    |    |    |    |    |
| B1 |    |    |    |    |    |    |    |    |
| B2 |    |    |    |    |    |    |    |    |
| B3 |    |    |    |    |    |    |    |    |
| B4 |    |    |    |    |    |    |    |    |
| B5 |    |    |    |    |    |    |    |    |
| B6 |    |    |    |    |    |    |    |    |
| B7 |    |    |    |    |    |    |    |    |
| B8 |    |    |    |    |    |    |    |    |
| B9 |    |    |    |    |    |    |    |    |
| B10|    |    |    |    |    |    |    |    |
| B11|    |    |    |    |    |    |    |    |
| B12|    |    |    |    |    |    |    |    |

Methodology: MWAF /13

- 1-2 Randomized Design
- 3-4 Mean square Design
- 5-6 SQR Design
- 7-8 QR Design
- 9-10 SSQR Design
- 11-12 SSQR Design

Statistical Analysis /1

**Step 1. Selecting Participants**
- There were 12 participants in this study, all of whom had adequate knowledge and experience in software development and MAS (min 4 years software development experience; passed AOSE course; designed and developed at least one MAS)
- The participants were provided sufficient documentation about the methodologies, clear instructions about the experiment, and equal amount of time to complete their task using well-prepared surveying questionnaire

Statistical Analysis /2

**Step 2. Selecting the appropriate statistical model**
- Complete Random Design (CRD)
  - Requires 36 participants to evaluate the 9 methodologies; each methodology will be evaluated by 4 subjects.
- Randomized Complete Block Design (RCBD)
  - Each evaluator should assess a complete block, i.e. 9 methodologies, and the design will probably more effective because we will have more replications, 12×9=108 instead of 36 (i.e. 9 replica/treatment).
  - Hard to implement; not all the participants are familiar with the whole set of methodologies.

** We decided to have at least 4 replica **
The statistical approach we adopted in MWAF depends on applying the Analysis of Variance (ANOVA) procedure to the Balanced Incomplete Block Design (BIBD) model to test for the significant differences in the mean effectiveness of each individual dimension among the evaluated methodologies.

### Statistical Analysis: BIBD /1

**Balanced Incomplete Block Design (BIBD)**

*the 12 participants have been used to evaluate the 9 methodologies, such that each participant will evaluate 3 methodologies only*

- observed rate of effectiveness, \( Y_{ij} = \mu + \alpha_i + \beta_j + \epsilon_{ij} \)
  - \( \mu \): overall mean of experiments
  - \( \alpha_i \): treatment main effects, the deviation from the mean caused by the \( i \)th treatment; \( \Sigma \alpha_i = 0 \) (fixed effects)
  - \( \beta_j \): main effects of the blocking variable, the deviation from the mean caused by the \( j \)th block; \( \Sigma \beta = 0 \) (fixed effects)
  - \( \epsilon_{ij} \): independent random error \( \sim N(0, \sigma^2) \)

### Statistical Analysis: BIBD /2

- **Incomplete**: each participant will not evaluate a complete set of methodologies
- **Balanced**: each block will have the same number of methodologies
- **Additive**: no interactions are considered between blocks and treatments
- **Fixed**: we were limited to narrow down the selection of the qualified methodologies upon our own interest, and not at random from a large number of methodologies

### Treatment Assignment

<table>
<thead>
<tr>
<th>Block (Participants)</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M7</th>
<th>M8</th>
<th>M9</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td>M2</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td>M3</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td>M4</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td>M5</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td>M6</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td>M7</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td>M8</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td>M9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
</tbody>
</table>

### Testing Model Aptness

<table>
<thead>
<tr>
<th>Test #</th>
<th>Test Type</th>
<th>Instrument(s) Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Outliers</td>
<td>a. Normal probability plot of residuals</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b. Individual value plot of residuals versus independent variable</td>
</tr>
<tr>
<td>2</td>
<td>Normality of residuals</td>
<td>Normal probability plot of residuals</td>
</tr>
<tr>
<td>3</td>
<td>Homogeneity of error variances</td>
<td>a. Residual plots against fitted values</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b. Bartlett’s test</td>
</tr>
</tbody>
</table>
There was no indication of non-normality, nor was there any evidence pointing to possible outliers: all the residuals fall within the boundaries of the 95% confidence interval.

**Probability Plot of Standardized Residual**

- Normal - 95% CI

Residuals seem to be unstructured and unrelated to the predicted values of the response variable, that is, no evidence of heterogeneity among the error variances.

**Residuals Versus the Fitted Values**

- Response is Rate

**Model Conclusion**

- We repeated these aptness tests for all 6 dimensions
- All of the plots for all dimensions suggest that ANOVA model is appropriate for the available data

**What is next? (two step evaluation)**

1. Evaluate each dimension
2. Pairwise comparison

**Statistical Hypotheses**

- **Null hypothesis H₀**
  - There is no significant difference in the mean effectiveness (τ₁) of the examined dimension among the evaluated AOSE methodologies, i.e., all the treatment effects are the same

- **Alternative hypothesis Hₐ**
  - There is a significant difference in the mean effectiveness (τ₁) of the examined dimension among the evaluated AOSE methodologies

**The null hypotheses test will allow to check whether there is a significant difference in the effectiveness of a specific dimension among the evaluated methodologies**

*If no variability observed, no need to continue the experiment*

However, if the test result is significant, we will go further to carry out multiple pairwise comparisons using Tukey’s HSD (honestly significant diffs!) method to identify which pairs of methodologies are significantly different from the others.
Testing Null Hypothesis

First conclusion:
- The criteria represented by Dimensions 1-6, as evaluated by their rate of effectiveness has shown significant differences among the 9 evaluated AOSE methodologies.
- Therefore pairwise comparison is conducted for all dimensions.

Evaluation Results /1

Indicates that our subjects could not reveal significant differences among the methodologies for this dimension!

Evaluation Results /2

Evaluation Results /3

Second conclusion: Methodologies ranked according to estimated mean of effectiveness of each dimension:

<table>
<thead>
<tr>
<th>Rank</th>
<th>Methodology</th>
<th>Est. Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M1: Gain</td>
<td>6.494</td>
</tr>
<tr>
<td>2</td>
<td>M2: MsSE</td>
<td>5.861</td>
</tr>
<tr>
<td>3</td>
<td>M3: Tropos</td>
<td>5.839</td>
</tr>
<tr>
<td>4</td>
<td>M4: Agent-Site</td>
<td>5.192</td>
</tr>
<tr>
<td>5</td>
<td>M5: PROMISE</td>
<td>5.049</td>
</tr>
<tr>
<td>6</td>
<td>M6: PROMISE</td>
<td>4.894</td>
</tr>
</tbody>
</table>

Evaluation Results /4

Evaluation Results /5
Evaluation Results

<table>
<thead>
<tr>
<th>Dimension 5: Application</th>
<th>Dimension 6: User Perception</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2: M@SE</td>
<td>M2: M@SE</td>
</tr>
<tr>
<td>M1: Guia</td>
<td>M1: Guia</td>
</tr>
<tr>
<td>M9: PASSI</td>
<td>M9: PASSI</td>
</tr>
<tr>
<td>M7: MESSAGE</td>
<td>M7: MESSAGE</td>
</tr>
<tr>
<td>M5: MASSIVE</td>
<td>M5: MASSIVE</td>
</tr>
<tr>
<td>M8: M@SE</td>
<td>M8: CommonKADS</td>
</tr>
<tr>
<td>M3: Tropos</td>
<td>M4: Agent-SE</td>
</tr>
<tr>
<td>M4: Agent-SE</td>
<td>M3: Tropos</td>
</tr>
<tr>
<td>M6: Prometheus</td>
<td>M6: Prometheus</td>
</tr>
</tbody>
</table>

Ranking Summary

- It is rather hard to devise an overall ranking for the evaluated methodologies (e.g., weighted averaging, linear optimization, etc.).

Recommended Future Works

1. Unifying AOSE methodologies

- The comparative statistical analysis we carried out in this research can actually be considered as a preliminary step towards unifying AOSE methodologies.
- Attributes receiving higher rate and weight are to be supported in the unified AOSE methodology.

2. Proposing agent-based software metrics

- Later we can evaluate AOSE methodologies to check whether they can deliver standardized scores that can be generalized as agent-based software metrics, or bench marks.
- This can be useful to evaluate any AOSE methodology by comparing the rates it gives against these metrics. That is, the relative efficiency of the methodology as compared to the AOSE metrics.

Limitations of Research

- Although the evaluation methodology is reliable and the model aptness tests were successful, the results obtained from this research are of "limited generality" mainly due to:
  - Type of experiment (i.e., balanced incomplete block design BIBD vs. complete block design CBD, etc.);
  - Limited resources of selecting experimental subjects from a relatively small population.
Conclusions

- We exhibited a statistical approach, based on adopting the BIBD model to evaluate the most common AOSE methodologies.
- We identified the most common and important criteria, upon which we evaluate an AOSE methodology.
- By implementing this approach, we come up with a set of significantly tested scores, in terms of population mean responses, that can be determined by applying multiple comparisons, after assuring significant differences among the methodologies.

References /1


References /2


The End!