



# PUC

ISSN 0103-9741

Monografias em Ciência da Computação  
nº 21/07

## **AUML-BP: A Basic Agent Oriented Software Development Process Model Using AUML**

**Maíra Athanázio de Cerqueira Gatti  
Arndt von Staa  
Carlos José Pereira de Lucena**

Departamento de Informática

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO  
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900  
RIO DE JANEIRO - BRASIL**

## AUML-BP: A Basic Agent Oriented Software Development Process Model Using AUML

Maíra Athanázio de Cerqueira Gatti, Arndt von Staa and Carlos José Pereira de  
Lucena

Laboratório de Engenharia de Software – LES  
Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brasil

{mgatti, arndt, lucena}@inf.puc-rio.br

**Abstract.** Agent-Oriented Software Engineering (AOSE) has emerged as the discipline devoted to the engineering of complex software systems based on the multi-agent systems paradigm. Research in the field of AOSE includes the identification and development of both conceptual tools (e.g., formal modeling) and practical tools (e.g., agent-based infrastructures) to support software engineers and programmers in the analysis, design and development of multi-agent systems. Among others, a great deal of effort in the AOSE area focuses on the definition of methodologies to guide the process of developing multi-agent systems. As the AOSE methodologies have been proposed so far, they are not enough for practical agent software development without a clear understanding of the software development process model that should underlie the methodology. In order to have a good process and successfully finish the project, it is necessary to explicitly adopt either general methods and methodologies, or specifically suitable ones. In this context, this paper proposes AUML-BP (AUML Basic Process), a basic agent oriented software development process model using AUML.

**Keywords:** Multi-Agent Systems; Software Engineering for Multi-Agent Systems, Modeling, Software Process, Software Development Process Model.

**Resumo.** A Engenharia de Software Orientada a Agentes (AOSE) emergiu como uma disciplina voltada para a engenharia de sistemas de software complexos baseados no paradigma de sistemas multiagentes. Para apoiar engenheiros de software e programadores na análise e desenvolvimento de sistemas multiagentes, pesquisadores da área vêm propondo ferramentas conceituais para a identificação e o desenvolvimento, tais como modelagem formal, além de ferramentas práticas, tais como infra-estruturas de agentes. Metodologias AOSE, como vêm sendo propostas, estão muito distantes de um modelo de processo de desenvolvimento que deve ser a base para se utilizar uma metodologia. Para ter um bom processo e um projeto completado com sucesso, é necessário adotar tanto métodos gerais como metodologias específicas, ou especificar metodologias adequadas. Neste contexto, este trabalho apresenta AUML-BP (*AUML Basic Process*), um modelo de processo de desenvolvimento de software orientado a agente com o uso de AUML.

**Palavras-chave:** Sistemas Multiagentes, Engenharia de Software de Sistemas Multiagentes, Processo de Software, Modelo de Processo de Desenvolvimento de Software.

**In charge of publications:**

Rosane Teles Lins Castilho  
Assessoria de Biblioteca, Documentação e Informação  
PUC-Rio Departamento de Informática  
Rua Marquês de São Vicente, 225 - Gávea  
22451-900 Rio de Janeiro RJ Brasil  
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530  
E-mail: [bib-di@inf.puc-rio.br](mailto:bib-di@inf.puc-rio.br)

# Table of Contents

1 Introduction	1
2 Processes vs. Methodologies	2
3 Related Works	3
4 The AUML Overview	4
4.1 AUML Diagrams	4
4.2 AUML Extensions	8
5 AUML-BP: A Basic Process Using AUML	10
5.1 How the Process is Organized	10
5.2 The AUML-BP Iteration	20
6 Conclusions and Future Works	21
References	22

# 1 Introduction

Agent technology enables the realization of complex software systems characterized by situation awareness and intelligent behavior, a high degree of distribution, as well as mobility support. Agent technology has the potential to play a key role in enabling intelligent applications and services by improving automation of routine processes, and supporting the nomadic users with pro-active and intelligent assistance based on principles of adaptation and self-organization. Hence, agent technology can open the way to new application domains while supporting the integration of existing and new software, and make the development process for such applications easier and more flexible [41].

Agents are often deployed in environments in which they interact, and sometimes cooperate with other agents (including both, people and software) that have possibly conflicting aims. Such environments are known as multi-agent systems [42]. Moreover, agents and multi-agent systems, other than a technology, represent a new paradigm for the development of autonomous task-oriented software entities that interact with each other in a high-level way (e.g., via co-operation, coordination of activities, negotiations), leading to possibly very articulated organizations (e.g., teams, coalitions, markets, swarms) [23].

In this context, agent-oriented software engineering (AOSE) [24][25] has emerged as the discipline devoted to the engineering of complex software systems based on the multi-agent systems paradigm. Research in the field of AOSE includes the identification and development of both conceptual tools (e.g., formal modeling) and practical tools (e.g., agent-based infrastructures) to support software engineers and programmers in the analysis, design and development of multi-agent systems. Among others, a great deal of effort in the AOSE area focuses on the definition of methodologies to guide the process of developing multi-agent systems.

AOSE methodologies, as they have been proposed so far, mainly try to suggest a clean and disciplined approach to analyze, design and develop multi-agent systems, using specific methods and techniques. Unfortunately, this is far from being enough for practical software development without a clear understanding of the software development process model that should underlie the methodology.

Accordingly, in the development of software systems and of multi-agent systems, the identification of a suitable methodology cannot abstract from the identification of a specific model for the software development process [26]. Such a model should define how the different phases of software development should be organized and coordinated with each other, which activities engineers and developers have to undertake in each stage and when, which technologies and artifacts may be used for those activities, which products have to be expected for each stage, and which resources need to be involved in the phases of software production process. In other words, the software process model should guide the entire production effort and complement the guidelines identified by a specific methodology.

It is well known that the real process of software construction, if not controlled, can become a chaotic effort with a low probability of reaching the desired goal within fixed limits of time and budget. Therefore, when an AOSE methodology is proposed with a lack of attention to some process model, this lack may strongly undermine the practical applicability of a methodology.

Moreover, while some well known and documented process models make it possible to easily capture good experiences and to transfer them to other persons, others only aim at introducing a minimum level of control of the software development chaos, thus allowing a high level of reactivity to very dynamic situations. These differences in process models have a direct consequence: in order to have a good process and successfully complete the project, it is necessary to adopt either explicitly general methods and methodologies, or specifically suitable ones.

In order to fulfill those needs, this paper proposes AUML-BP (AUML Basic Process), a basic agent oriented software development process model using AUML, and is outlined as follows: Section 2 sums up the main related works of agent-oriented software development process, Section 3 presents the main AUML concepts, Section 4 describes the basic process for the agent-oriented software development proposed in this paper, and Section 5 presents the conclusions and future works.

## 2 Processes vs. Methodologies

Generally speaking, a **development process** (or simply process) is an ordered set of steps that encompasses all of the activities, constraints and resources required to produce a specific desired output (e.g., a physical artifact) satisfying a set of input requirements. Typically, a process is composed of different phases placed in relation to each other. Each phase of a process identifies a portion of work (more properly called work definition) to be conducted within the context of the process, the resources to be exploited to that purpose and the constraints to be obeyed in the execution of the phase. Phases are usually comprised of a set of activities that may, in turn, be conceived in terms of smaller atomic units of work (steps) [1].

**Software development process** (or simply software process) is the coherent set of policies, organizational structures, technologies, procedures and artifacts that are needed to conceive, develop, deploy and maintain (evolve) a software product [43]. Consequently, we also can identify that software processes are typically made up of a set of phases, each specifying which activities should be carried out and which roles (i.e.: client, analyst, software architect, programmers, etc.) and resources are to be involved in them. However, unlike traditional “development” processes, software processes should also take into account the fact that the product should not only be developed but also conceived, often relying on unstable or incomplete requirements; deployed, i.e., put to work in an operational environment; maintained and evolved, depending on novel requirements or changes in the operational environments [1].

On the other hand, a **software (development) process model** prescribes the phases around which a process should be organized, which activities should be executed in some of the phases, in which order such phases should be executed and when iterations and coordination between the work of the different phases should occur. In other words, a process model defines a skeleton, a template, around which to organize and detail an actual process. A software development process model (or simply a “process model”) does not take care of fine-grained work definitions, guidelines, modeling style for artifacts, as these can change and be adapted from case to case. This is one of the most important aspects of process models [1].

To be successfully applicable, any phase of a process should be complemented by some methodological guidelines (including the identification of the techniques and tools to be used, and the definition of how artifacts have to be produced) that could

help the involved stakeholders accomplish their work according to some defined best practices.

A methodology is a collection of methods covering and connecting different phases in a process. The purpose of a methodology is to prescribe a certain coherent approach to solving a problem in the context of a software process by pre-selecting a number of methods [44]. Just to clarify, a method prescribes a way of performing some kind of activity within a process, in order to properly produce a specific output (i.e., an artifact or a document) starting from a specific input (again, an artifact or a document).

### 3 Related Works

From the very beginning of software engineering research, a variety of software process models have been proposed, from sequential waterfall-like to evolutionary and transformation-based ones, with the goal of identifying effective, reliable and reproducible ways to produce software. In the software engineering community there is now a general consensus that for most real-world industrial projects the pervasive waterfall model should be replaced by more flexible and iterative approaches, such as evolutionary or spiral ones .

Also, it is an acknowledged fact that no single general-purpose process model can be effective for all projects, and that different commercial and engineering needs may be satisfied by different process models. In addition, software processes cannot be defined and established once and for ever; they need to be continuously assessed and improved.

Most of the AOSE methodologies as, for instance, Gaia [22], Roadmap [27], Prometheus [28], MASE [29], AOR [30], Massive [31], Ingenias [32], Tropos [33], (Agile) PASSI [4][5], etc., adopt either a waterfall-like or an evolutionary/incremental model [1]. In particular these methodologies do not make any explicit reference to the process model, ending up in promoting a rather standard waterfall process model or – more rarely – a rough incremental process model. The methodologies that pay more attention to the process model issue end up explicitly proposing an incremental process model.

Summarizing, we can state that the need for incremental process models is widely recognized in the community. Very few methodologies adopt a transformation-like model, such as, for instance, the DESIRE methodology. Although other attempts in transforming informal specification into code by means of a transformation process have been explored so far (consider e.g., Z schemas [34]), these efforts are to be considered single methods and notations more than complete methodologies.

Spiral models, too, have encountered very limited success. Very likely, the reason is that only a few complex industrial projects (involving high risks) so far have been carried out. Thus, the need to anticipate and possibly eliminate the risks associated with complex software development projects in agent based development have simply not emerged.

## 4 The AUML Overview

Agent Unified Modeling Language (AUML) [12] is a graphical modeling language that is being standardized by the Foundation for Intelligent Physical Agents (FIPA [20]) Modeling Technical Committee. AUML was proposed as an extension of the Unified Modeling Language (UML). So far, there is no recognized standard for modeling a MAS and AUML emerged as a candidate to assume such a position.

AUML uses decomposition, abstraction and organization characteristics, which are the attitudes for reducing the complexity of development software. AUML decomposes a system into small parts of objects, models, use-case or class, various operational actions. Concerning the abstraction, it provides a specialized abstract view of modeling (class, use-case, diagram, interface etc.). It is used to create a set of semantics and conditions for operation and infrastructure services.

The agent-oriented organization defines a range of elements and notations as a requirements specification for domain modeling. It aims to provide a model and an internal architecture of an agent system. It usually offers some frameworks (class, diagram, interface, etc.) to show how agents can be constructed in an agent system. The modeling focuses on one aspect at a time and increases the ability to understand complex problem issues during the time of system design.

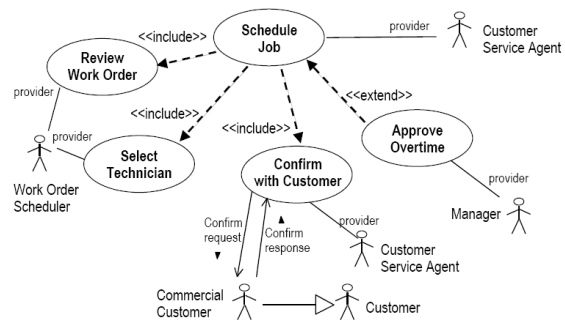
The core parts of AUML are mechanisms to model protocols for multi-agent interaction. This is achieved by introducing a new class of diagrams into UML: protocol diagrams. Protocol diagrams extend UML state and sequence diagrams in various ways. Particular extensions in this context include agent roles, multithreaded lifelines, extended message semantics, parameterized nested protocols, and protocol templates.

This section is based on the main articles and references of AUML, and further details can be found from [9] to [21].

### 4.1 AUML Diagrams

#### AUML Use Case Diagram

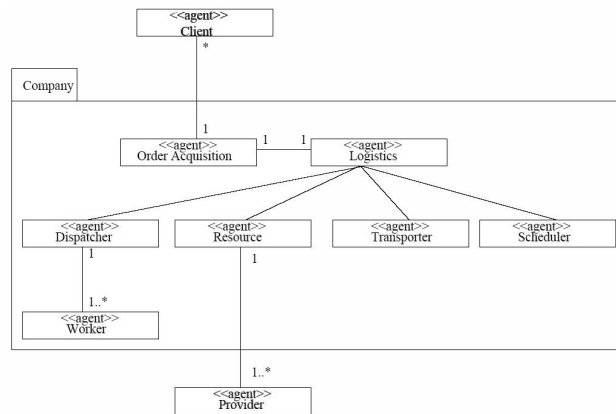
AUML Use Cases capture goal-oriented interactions between agents with specified roles and the software system. They are a set of usage paths through the system, each with a discrete goal.





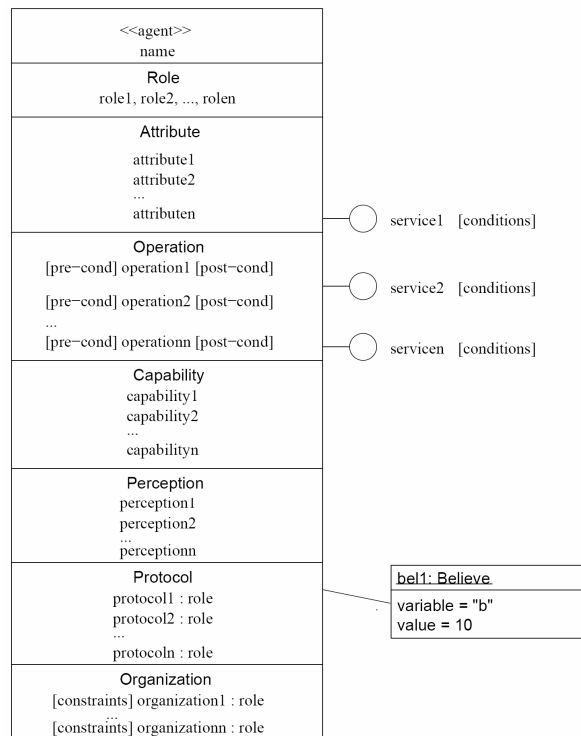
## AUML Class Diagram

AUML Class Diagram describes the types of agents in the system and their static relationships.

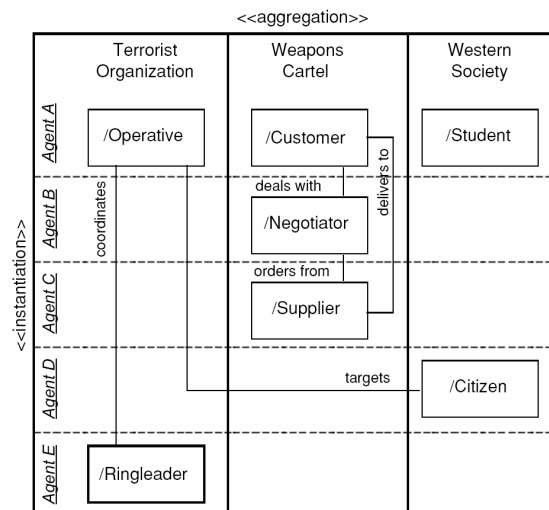


Considering the agent view, each agent class states its roles, attributes and operations.

The agent class also defines the capabilities of that agent in the organization, the perceptions in the environment, which basically are the sensors, the protocols on which the agent interacts with other agents, and the set of organizations where the agent plays the roles with their constraints.



Considering the organizational view, it is possible to describe the types of agents with their roles in the organization and their static relationships.

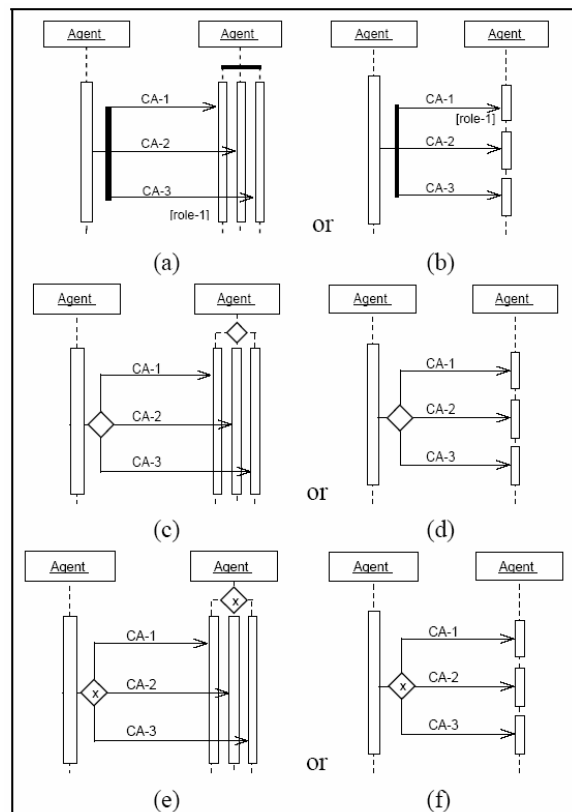


## AUML Sequence Diagram

The definition of an agent interaction protocol (AIP) describes a communication pattern, with an allowed sequence of messages between agents having different roles, constraints on the content of the messages, and a semantics that is consistent with the communicative acts (CAs) within a communication pattern.

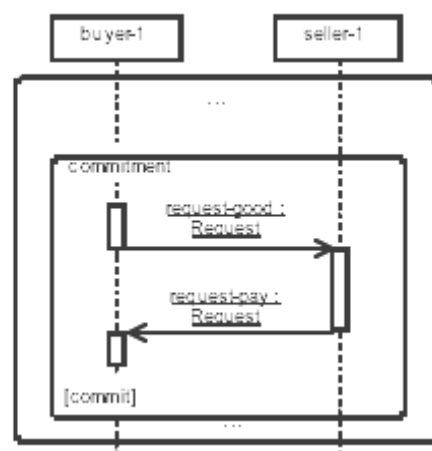
The lifeline may split up into two or more lifelines to show AND and OR parallelism and decisions, corresponding to branches in the message flow. Lifelines may merge at some subsequent point.

The XOR can be abbreviated by interrupting the threads of interaction. The thread of interaction, i.e. the processing of incoming messages, is split up into different threads of interaction; in the case of the behavior of an agent role it depends on the incoming message. The lifeline of an agent role is split accordingly and the thread of interaction defines the reaction to different kinds of received messages.

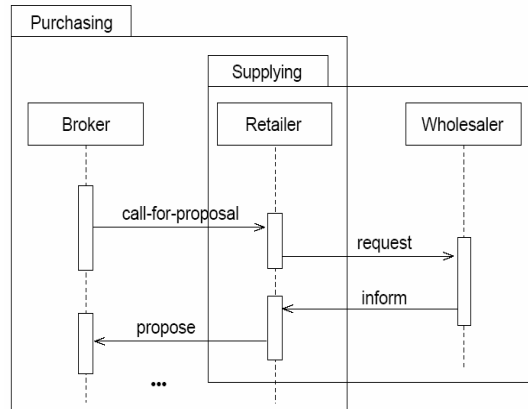


The purpose of protocol templates is to create reusable patterns for useful protocol instances. First, the protocol as a whole is treated as an entity in its own right. The protocol can be treated as a pattern that can be customized for other problem domains.

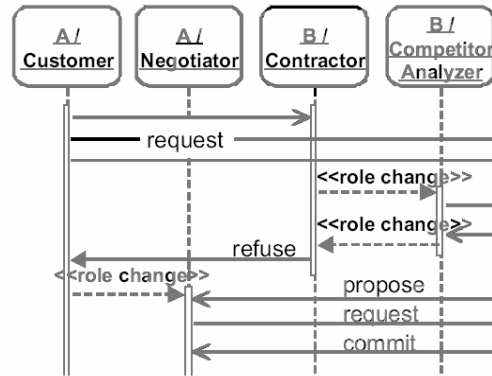
A nested protocol is a protocol within another protocol. Additionally nested protocols are used for the definition of repetition of a nested protocol according to guards and constrains.



An interleaved protocol is a protocol that needs a part (not complete) of another protocol to be completed.

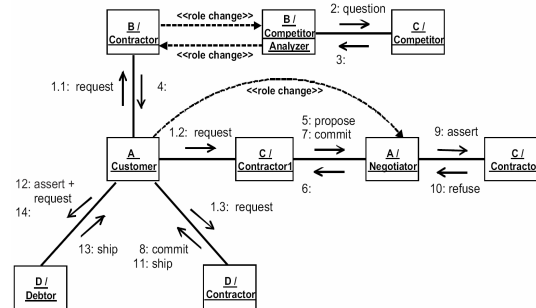


Agents can perform various roles within one interaction protocol.



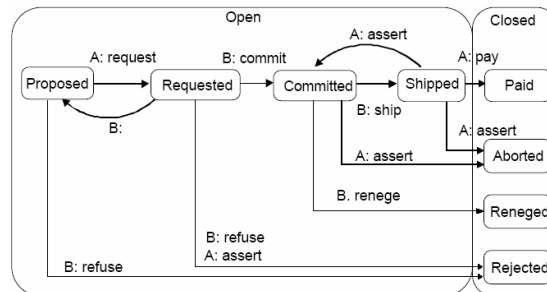
**AUML Collaboration Diagram**

It shows the dynamic interaction of the roles/ agents in a system. The messages are the performative exchanged among roles/ agents.



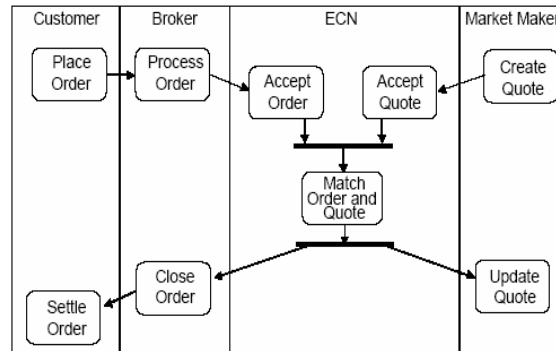
**AUML State Diagram**

These show the transitions and states of a protocol or of a role.



## AUML Activity Diagram

These show the activities of a protocol or of a role.



## 4.2 AUML Extensions

While conducting experiments with the AUML we discovered a gap between the AUML Use Cases and static and dynamic diagrams. Basically, it was difficult to depict the agent functionalities into protocols, state charts and activities. Hence we propose the use of Agent Stories and Agent Index Cards. They were inspired by the User Stories [46] present in XP [45], which actually are mini-user stories and “tasks” are stated in user-oriented terms. In our case, Agent Stories are mini-agent stories and agent tasks are stated in agent-oriented terms as used in autonomous processors. And the Agent Index Card is just a way of planning the agent story implementation.

An AUML Use Case has one or more Agent Stories depending on the number of agents that interact with it. An Agent Story is a scenario that in a separated way describes all the agent goals and tasks required to accomplish its goals. An agent may have one or more Agent Stories depending on the number of AUML Use Cases in which the agent participates.

In order to prioritize the Agent Stories implementation and distribute it in iterations, Agent Index Cards may be used. An Agent Index Card is a prioritized card that contains the following information: name, importance, notes, estimation time, and how to demo.

For instance, suppose a simple AUML Use Case on which an Agent A plays the customer role, and an Agent B plays the seller role. Agent A wants to order some books from Agent B.

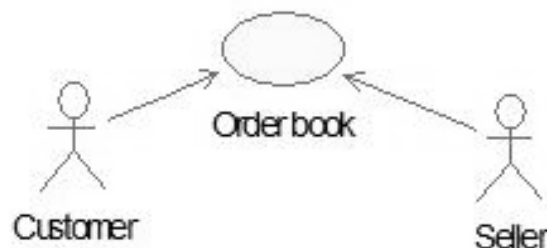


Figure 1 - AUML Use Case partial view

The AUML Use Case description partial view would be:

AUML Use Case	Order book
Roles	Customer Seller
Pre-conditions	The agent that plays the Customer role may not be the same as the one that plays the Seller role
Description	<ol style="list-style-type: none"> <li>1. The Customer sends a message to all the agents playing the Seller role asking for books.</li> <li>2. A Seller agent answers the Customer asking for information about the book.</li> <li>3. The Customer sends the name or other information.</li> <li>4. The Seller checks in the systems if the book is available. <ol style="list-style-type: none"> <li>a. If it is, the Seller sends a message to the Customer with the price.</li> <li>b. If it is not, the Seller sends a negative message.</li> </ol> </li> </ol> <p>....</p>

The Customer Agent Story for this AUML Use Case is:

Role	Customer
AUML Use Case	Order a book
Goal	Order a book
Tasks	<ol style="list-style-type: none"> <li>1. Send the message M1 to the Seller</li> <li>2. Wait for the Seller answer</li> <li>3. Receive the Seller answer</li> <li>4. If the Seller answer is positive and contains the price, analyze the price. <ol style="list-style-type: none"> <li>a. If can afford it, send the message M2.</li> <li>b. If cannot afford it, send the message M3.</li> </ol> </li> <li>5. If the Seller answer is negative, sent the message M4.</li> </ol>
Messages	<p>M1 : [performative: Inform; sender: Customer; receiver: Seller; content: book name]</p> <p>M2 : [performative: Reject; sender: Customer; receiver: Seller; content: the price is expensive]</p> <p>....</p>

Through the Agent Index Card set define in which order each Agent Story must be developed. It defines the order according to the Agent Index Card more interdependently than others and according to the more important agent functionalities. For instance, see the Agent Index Card below:

**Agent Story Order book**

**Notes**

Need a AUML sequence diagram and a collaboration diagram. No need to worry about activity diagram.

**How to demo**

Log in, open order book page, input the book name, go to ordered books page and check if there is any answer. If not, wait some time and refresh the page. Check if there is a positive answer with the bought book, or if there is a positive answer with a dependence, like more information to choose the book, or if there is a negative answer.

**Importance**

8

**Estimate**

12hr

**Figure 2- Agent Index Card for the Order Book Agent Story**

## 5 AUML-BP: A Basic Process Using AUML

AUML-BP is a software development process model that is expected to cover a broad set of agent development needs. AUML-BP combines OpenUP/Basic [7][35] with the AUML model language. OpenUP/Basic takes an agile approach to software development, with only fundamental content providing a simplified set of work products, roles, tasks and guidance. It is an iterative software development process that is minimal, complete and extensible. It is a process for small, co-located teams that value collaboration and stakeholder benefits over unnecessary deliverables and formality.

From the descriptions of methodology processes we will extract the method fragments. A method fragment is a reusable part of a design process that takes some already designed pieces of the system and produces a new part of the design following a precise procedure. The FIPA method fragment definition [36] is composed as follows:

1. A portion of process
2. One or more deliverables (artifacts like (A)UML/UML diagrams, text documents and so on). Some preconditions (like the required input data or guard condition)
3. A list of concepts (related to the MAS meta-model) to be defined/designed/refined by executing the specific method fragment.
4. Guideline(s) that illustrate(s) how to apply the fragment and best practices related to it. A glossary of terms used in the fragment
5. Other information (composition guidelines, platform to be used, application area and dependency relationships useful to assemble fragments) completes this definition.

### 5.1 How the Process is Organized

AUML-BP method content is focused on a subset of RUP [37] disciplines as follows: requirements, analysis & design, implementation and test (Figure 3). Although the implementation discipline is based on Test-Driven Development (TDD) [38] which is a software development technique that involves repeatedly first writing a test case and then implementing only the code necessary to pass the test. It is our belief that TDD is an effective way of implementing MAS and some primary result efforts in this direction can be found in [47].

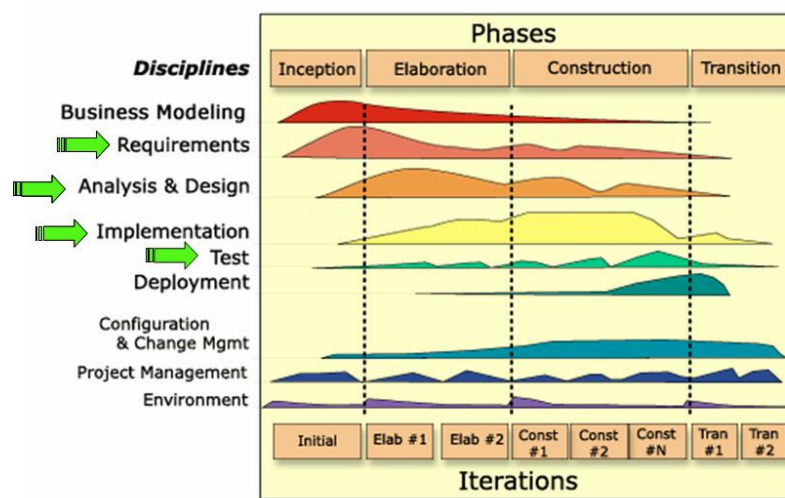
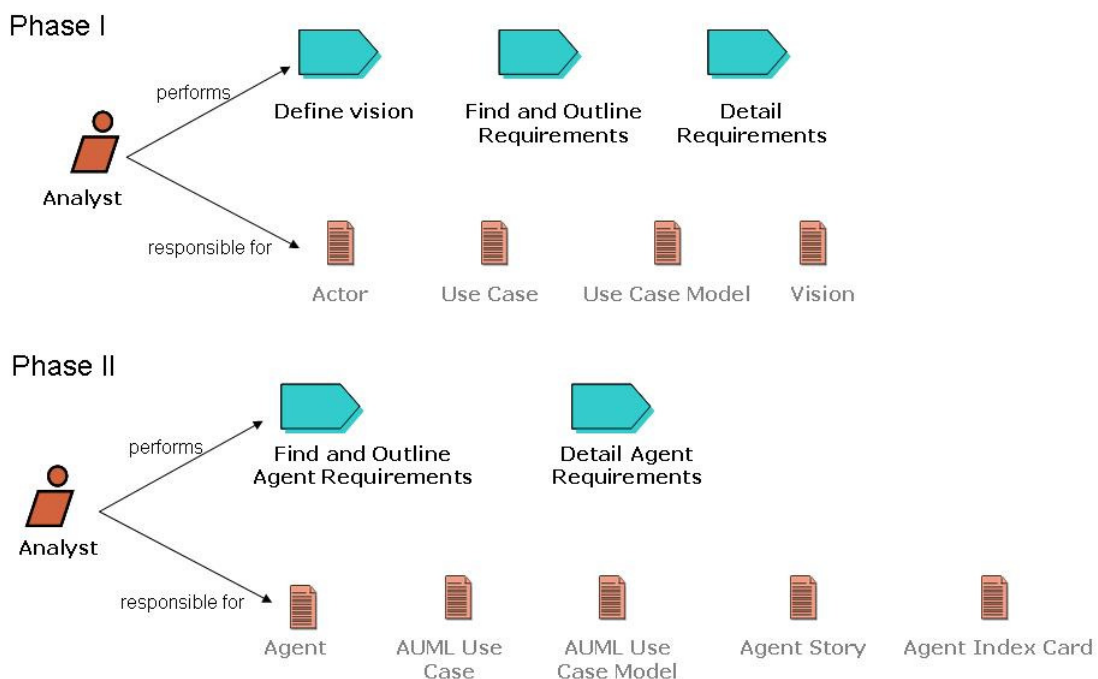



Figure 3 - The RUP Phases and Disciplines, adapted (copied) from [37].

Analysis & design and Test content are not called only in a separate discipline. The developer performs low level analysis and design, by identifying classes and internal parts of components. Implementation discipline concentrates tasks the developer performs to evolve the design into implementation, which is unit-tested and integrated into the code base.


In what follows we describe for each role the agent related tasks to perform and the agent related artifacts it is responsible for when executing the activities.

□ **Analyst** - responsible for gathering agent requirements and documenting them as needed. The Analyst is also responsible for designing the agent solution.

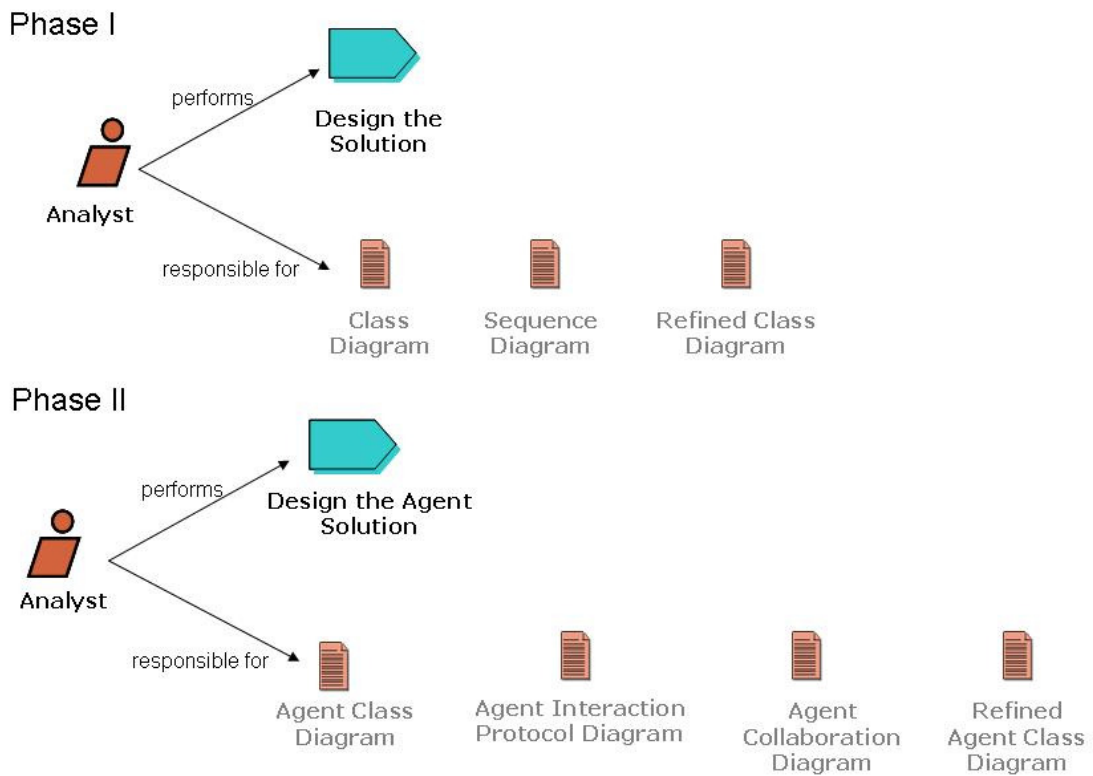



 Task	<b>Find and Outline Agent Requirements</b>
Description	This task describes how to capture the agent requirements for the system.
Purpose	The purpose of this task is to understand stakeholder requirements considering the agent goals in the systems and communicate these to the development team.
Discipline	Requirements
Role	Analyst
Input	Glossary Vision Supporting Requirements Use Case Use Case Model

Output	Supporting Agent Requirements Agent AUML Use Case AUML Use Case Model
Guidelines	Agent Requirements Gathering Techniques Find and Outline Agents and AUML Use Cases Supporting Agent Requirements

 Task	<b>Detail Agent Requirements</b>
Description	This task describes how to detail one or more agent requirements for the system.
Purpose	The purpose of this task is to describe one or more agent requirements in sufficient detail to validate understanding of the agent requirement, to ensure concurrence with stakeholder expectations and to permit software development to begin.
Discipline	Requirements
Role	Analyst
Input	Glossary Vision Supporting Agent Requirements Agent AUML Use Case AUML Use Case Model
Output	Agent AUML Use Case AUML Use Case Model Agent Story Agent Index Card
Guidelines	Detail AUML Use Case and Scenarios Create and Detail Agent Story Create and Detail Agent Index Card AUML Use Case Formats Agent Story Formats Agent Index Card Formats

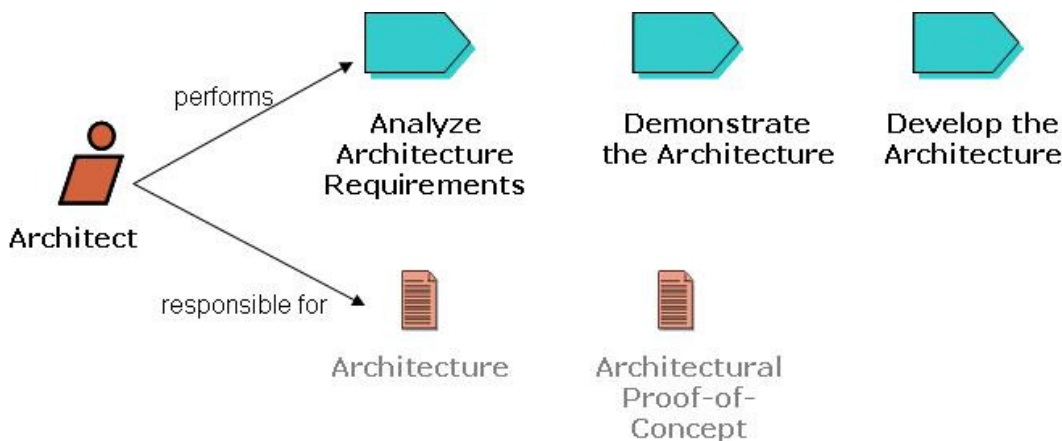





 Task	<b>Design the Agent Solution</b>
Description	Identify the elements and devise the agent interactions, behavior, relations and data necessary to realize some functionality. Render the agent design visually to aid in solving the problem and communicating the solution.
Purpose	The purpose of this task is to describe the software agents so that they support the required behavior, are of high quality and fit within the architecture.
Discipline	Analysis & Design
Role	Analyst
Input	Class Diagram Sequence Diagram Architecture Supporting Agent Requirements AUML Use Case Agent Story Agent Index Card
Output	Agent Class Diagram Agent Interaction Protocol Diagram Agent Collaboration Diagram


	(Refined) Agent Class Diagram
Guidelines	Agent Design AUML Use Case Realizations Agent Communication Patterns Agent Designing Visually


□ **Architect** - responsible for the software architecture, which includes the key technical decisions that constrain the overall design and implementation for the project. It also includes the agent software architecture and its features.



 Task	<b>Analyze Architecture Requirements</b>
Description	Analyze the architecturally significant requirements and define an architecture candidate for the system based on experience gained from similar systems or in similar problem domains. Define the architecture patterns, key mechanisms, and, where applicable, modeling conventions for the system.  As a second step, apply the same steps to the agent architecture requirements.
Purpose	To provide sufficient guidance and direction for the team to be able to perform analysis and design in consistent and coherent ways.
Discipline	Analysis & Design
Role	Architect
Input	Glossary Vision Use Case Model AUML Use Case Model Agent Story
Output	Architecture

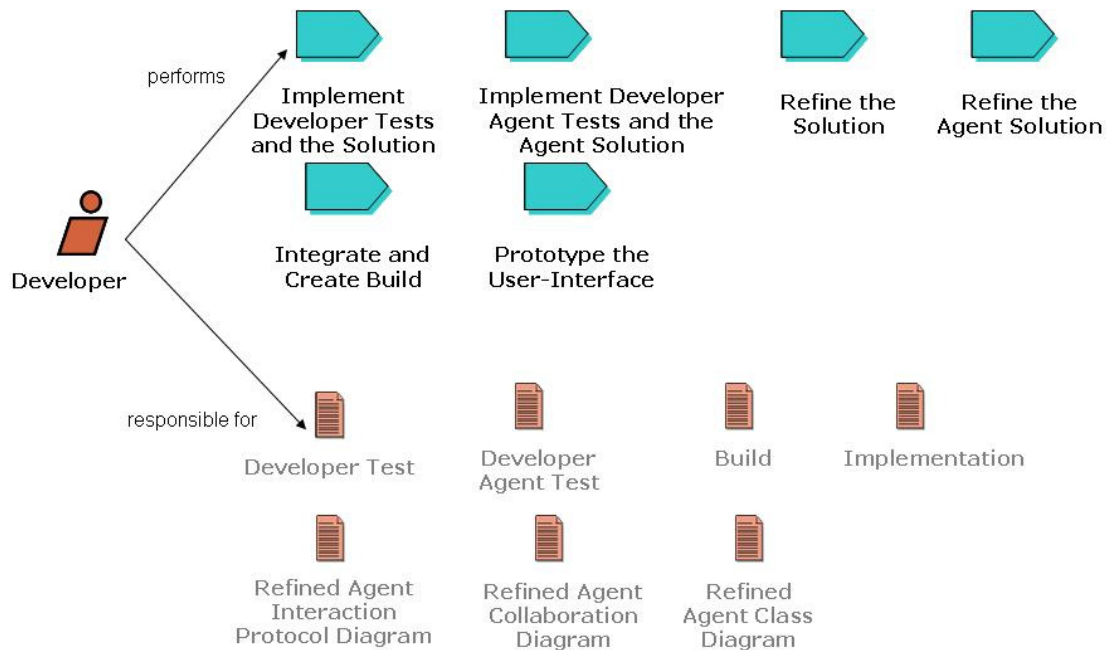
	Agent Architecture Design Agent Design
Guideline	Analyze the Architecture Analyze the Agent Architecture Analyze the Integration of both Architectures


 Task	<b>Demonstrate the Architecture</b>
Description	Present at least one solution that proves that the planned (agent) architecture will meet the agent requirements.
Purpose	Reduce the risk of reworking the software (agent) architecture by illustrating at least one architecture that supports the (agent) requirements of the system.
Discipline	Analysis & Design
Role	Architect
Input	Vision Supporting Requirements Architecture Supporting Agent Requirements Agent Architecture
Output	Architectural Proof-of-Concept Agent Architectural Proof-of-Concept
Guidelines	Architectural Proof-of-Concept Agent Architectural Proof-of-Concept

 Task	<b>Develop the Architecture</b>
Description	Make concrete decisions about the (agent) architecture to provide guidance and direction to the development work for the iteration.
Purpose	Provide a skeletal design with the agent skeletal design also to enable more comprehensive design activities to be performed coherently by the team.
Discipline	Analysis & Design
Role	Architect
Input	Vision Supporting Requirements


	Architecture Supporting Agent Requirements Agent Architecture Design Agent Design
Output	Architecture Design Agent Architecture Agent Design
Guidelines	Identify design mechanisms Identify reuse opportunities Identify architecturally significant design elements Define development and test architectures Document and communicate decisions

□ **Developer** - create a solution (or part of it) by doing (agent) design, (agent) implementation, unit tests and integration of components and agents.





 Task	<b>Implement Developer Tests and the Solution</b>
Description	Implement one or more tests that enable the validation of the individual software components through execution. Implement source code necessary to pass the test and to provide new functionality or fix defects.

Purpose	<p>The purpose of this task is to produce an implementation for part of the solution (such as a class or component), or to fix one or more defects. The result is typically new or modified source code, which is generally referred to the implementation.</p> <p>Test-Driven Development (TDD) is a software development technique that involves repeatedly first writing a test case and then implementing only the code necessary to pass the test.</p>
Discipline	Implementation & Test
Role	Developer
Input	<p>Supporting Requirements</p> <p>Use Case</p> <p>Design</p>
Output	Implementation
Guidelines	<p>Test-Driven Development (TDD)</p> <p>Mock Objects</p> <p>Refactoring</p>

 Task	<b>Implement Developer Agent Tests and the Agent Solution</b>
Description	<p>Implement one or more tests that enable the validation of the individual software agents through execution.</p> <p>Implement source code necessary to pass the test and to provide new agent functionality or fix agent defects.</p>
Purpose	The purpose of this task is to produce an implementation for part of the agent solution (such as a protocol or component), or to fix one or more agent defects. The result is typically new or modified source code, which is generally referred to the agent implementation.
Discipline	Implementation & Test
Role	Developer
Input	<p>Supporting Requirements</p> <p>Supporting Agent Requirements</p> <p>Use Case</p> <p>Agent Use Case</p> <p>Design</p> <p>Agent Design</p>
Output	<p>Implementation</p> <p>Agent Design</p>
Guidelines	<p>Test-Driven Development (TDD)</p> <p>Mock Agents</p>

	Refactoring
--	-------------

 Task	<b>Refine the Solution</b>
Description	Identify the elements that had their interactions, behavior, relations, and data necessary to realize some functionality updated/ refactored.
Purpose	The purpose of this task is to keep the design consistent.
Discipline	Implementation & Test
Role	Developer
Input	Design (Class Diagram, Sequence Diagram)
Output	Design (Class Diagram, Sequence Diagram)
Guidelines	Refactoring

 Task	<b>Refine the Agent Solution</b>
Description	Identify the agents that had their interactions, behavior, relations and data necessary to realize some functionality updated/ refactored.
Purpose	The purpose of this task is to keep the agent design consistent.
Discipline	Implementation & Test
Role	Developer
Input	Agent Design Agent Class Diagram Agent Interaction Protocol Diagram Agent Collaboration Diagram
Output	Agent Design Agent Class Diagram Agent Interaction Protocol Diagram Agent Collaboration Diagram
Guidelines	Refactoring

The Tester role has the same activities as in the OpenUP/Basic – responsible for testing the system from a larger perspective than the developer does, making sure the system works as defined and is accepted by the customer. Hence this role will not be described here since it can be found in the OpenUP/Basic definition [35].

The guidelines explain how to informally represent the artifacts. In general, these guidelines recommend capturing the information in an existing artifact, spreadsheet, database, table, e-mail, etc.

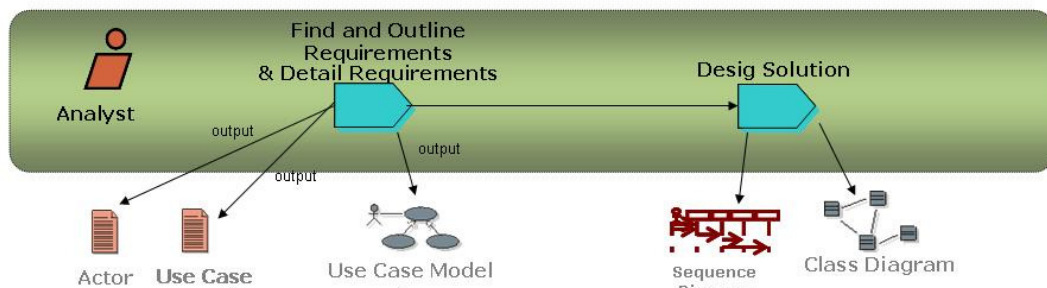
Reusable method content is created separate from its application in processes. Method content provides step-by-step explanations, describing how specific development goals are achieved independent of the placement of these steps within a development life cycle.

Processes take method elements (step-by-step explanations, describing how specific development goals are achieved) and relate them into semi-ordered sequences that are customized to specific types of projects. In OpenUP/Basic the method's elements are organized into reusable pieces of process called capability patterns, providing a consistent development approach to common problems. These patterns are made of activities organizing tasks (from the method content), grouping them in a sequence that makes sense for the particular area where that pattern is applied.

Moreover OpenUP/Basic has a delivery process for iterative development throughout four phases. The iteration template patterns are put together, as many times as needed, depending on how the project manager needs to instantiate them to create a project plan.

The figures below summarize and illustrate how a set of development goals are related and achieved step-by-step:

Phase I



Phase II

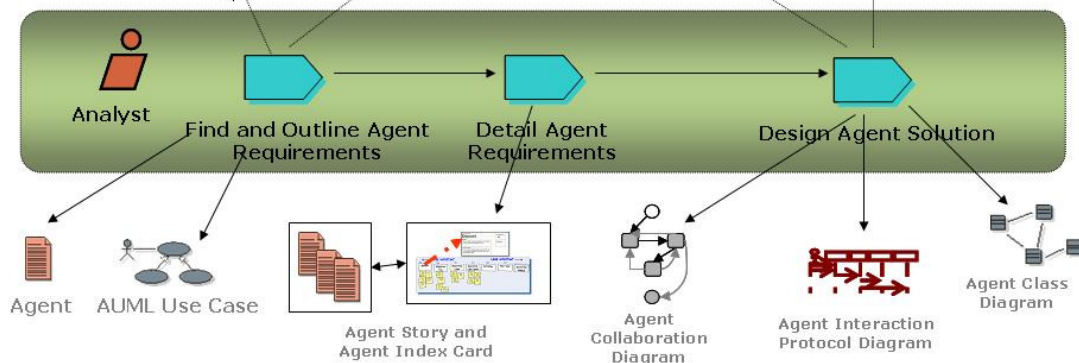


Figure 4 - The Analyst activities and artifacts relationships step-by-step

For instance, when the Analyst executes the Find and Outline Agent Requirements he/she uses the Actor and Use Case artifacts in order to refine it and extract the agent capabilities, functionalities and interactions. The result of this activity will be the Agent and AUML Use Case artifacts.

The Detail Agent Requirements activity has the Agent and AUML Use Case artifacts as the inputs and generates the Agent Stories and Agent index Cards artifacts outputs.

Finally, the Design Agent Solution activity receives Agent, AUML Use Case, Agent Stories and Agent Index Cards as input and generates the Agent Collaboration, Sequence and Class Diagrams which will be refined by the Developer in the Refine the

Agent Solution activity after the execution of the Implement Developer Agent Tests and the Agent Solution activity as shown in Figure 5 below.

The Developer may use Mock Agents [39] when executing the Implement Developer Agent Tests and the Agent Solution activity. Mock Agent would be used as agent unit test based on the Agent Stories and Agent Index Card, and the goal is to develop the agent to pass in the agent unit test.

After all the agents are implemented separately, then it is necessary to execute integration tests, which will be the execution of test cases based on the AUML Use Case. The integration test represents the real agent interactions and the goal is to find any logical path fault during their executions, for instance a deadlock.

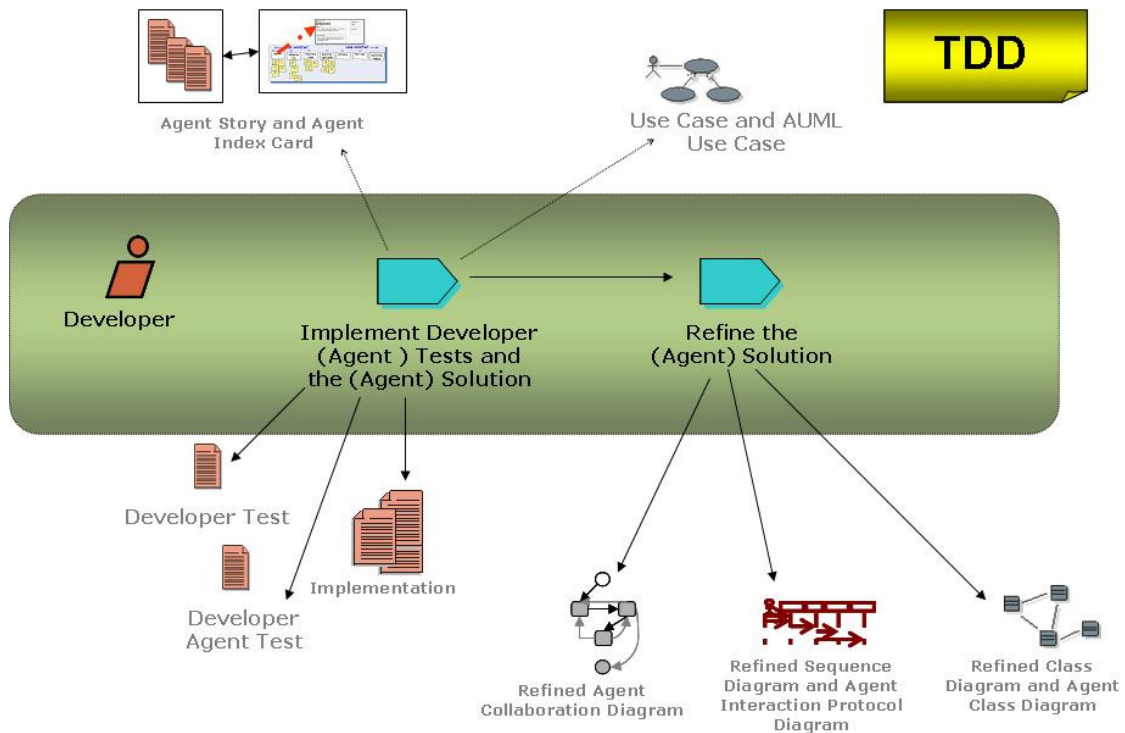


Figure 5 - The Developer activities and artifacts relationships step-by-step using Test Driven Development

## 5.2 The AUML-BP Iteration

As stated previously, AUML-BP uses iterative development with a life cycle consisting of several iterations, just like RUP. Iteration incorporates a loosely sequential set of activities in business modeling, requirements, analysis and design, development (which means implementation & test), test and deployment, in various proportions depending on where in the development cycle the iteration is located.

The project is broken down into four phases: Inception (deciding what to build), Elaboration (addressing the largest risks, demonstrate technical feasibility), Construction (building the software with a working version at each iteration), Transition (documentation, training, deployment of software).

Each phase is made up of iterations. This allows an evolving understanding of the agent requirements, continuous user involvement and addresses the highest risks first.



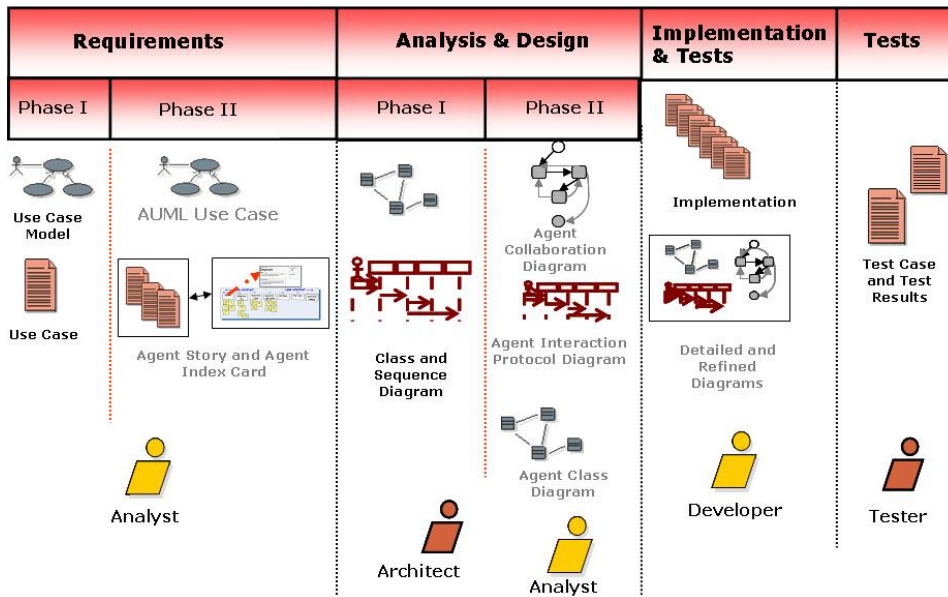


Figure 6 - An AUML-BP Iteration

## 6 Conclusions and Future Works

This work presented AUML-BP, a basic process for the development of agent-oriented systems using the AUML modeling language. In order to define the process, it was necessary to extend the AUML since it does not define any methods for the integration of the requirements phase with the analysis & design phase, which is important during agent software development because the agent sequence, agent collaboration and agent activity diagram designs are not feasible tasks. Thus we proposed the use of Agent Story and Agent Index Card for this purpose.

We described all the activities with their respective inputs and outputs and their associated roles. And we also illustrated the process with the main agent related activities step-by-step and an AUML-BP iteration. We still have to evaluate this process with a case study.

We also want to create an EPF Composer plug-in for the AUML-BP such as there is for the OpenUP/Basic and others. The EPF Composer (Eclipse Process Framework Composer) [40] is a tool platform for process engineers, project leads, project and program managers who are responsible for maintaining and implementing processes for development organizations or individual projects. With EPF, it is possible to keep and maintain a knowledge base of intellectual capital that allows us to browse, manage and deploy content.

EPF Composer also provides catalogs of pre-defined processes for typical project situations that can be adapted to individual needs. It provides a way of representing best development practices for specific disciplines, technologies or development styles, and allows you to set-up your own organization-specific capability pattern libraries. Finally, the documented processes created with EPF Composer can be published and deployed as Web sites.

Thus we believe that the AUML-BP plug-in for EPF Composer can be used as a powerful tool during the agent-oriented development process.

## References

- [1] L. Cernuzzi, M. Cossentino, F. Zambonelli. **Process Models for Agent-Based Development**. *International Journal on Engineering Applications of Artificial Intelligence (EAAI)*. Elsevier. 2004.
- [2] T. De Wolf, and T. Holvoet, **Towards a Methodolgy for Engineering Self-Organising Emergent Systems**, In *Self-Organization and Autonomic Informatics (I)*, Vol. 135 of *Frontiers in Artificial Intelligence and Applications*. H. Czap, R. Unland, C. Branki and H. Tianfield (editors), pp 18 - 34. ISBN: 1-58603-577-0, IOS Press. Proc. of the Int. Conf. on Self-Organization and Adaptation of Multi-agent and Grid Systems (SOAS 2005), Glasgow, Scotland, UK (PDF) [Best Paper Award]
- [3] **PASSI: a Process for Agents Societies Specification and Implementation** ([web site](#))
- [4] A. Chella, M. Cossentino, L. Sabatucci, and V. Seidita. **From PASSI to Agile PASSI: Tailoring a Design Process to Meet New Needs**. In Proc. of IEEE/WIC/ACM International Joint Conference on Intelligent Agent Technology, Beijing, China, September 2004.
- [5] A. Chella, M. Cossentino, L. Sabatucci, and V. Seidita. **Agile PASSI: An Agile Process for Designing Agents**. *International Journal of Computer Systems Science & Engineering*. Special issue on "Software Engineering for Multi-Agent Systems," 21(2). March 2006.
- [6] M.Cossentino and A. Chella. **Designing a problem specific design process for multi-agent systems**. In Proc. of The Intelligent Agent Architectures: Combining the Strengths of Software Engineering and Cognitive Systems Workshop at the Nineteenth National Conference on Artificial Intelligence, San Jose, California, July 2004.
- [7] *Basic Unified Process: A Process for Small and Agile Projects* <http://www.eclipse.org/proposals/beacon/Basic%20Unified%20Process.pdf>, July, 2007.
- [8] **Scrum: A Simple Process For Managing Complex Projects** <http://www.controlchaos.com/>, July, 2007.
- [9] Odell, J., Parunak, H. and Bauer, B. (2000) "**Extending UML for Agents**," In: Wagner, G., Lesperance, Y. and Yu, E. Proceedings of the Agent-Oriented Information Systems Workshop, AOIS 2000, Eds., Austin, pp. 3-17.
- [10] Odell, J., Parunak, H. and Bauer, B. (2001) "**Representing agent interaction protocols in UML**," In *Agent-Oriented Software Engineering, First International Workshop, AOSE 2000*, Ciancarini, P., Wooldridge, M., Eds., LNCS 1957 Springer, Limerick, Ireland. pp. 121-140.
- [11] Parunak, H. and Odell, J. (2002), "**Representing social structures in UML**," In *Agent-Oriented Software Engineering II*, Wooldridge, M., Weiss, G. and Ciancarini, P., Eds., LNCS 2222, Springer-Verlag, Berlin, pp. 1-16.

- [12] B. Bauer, J. P. Müller, J. Odell. **Agent UML: A Formalism for Specifying Multiagent Interaction**. Agent-Oriented Software Engineering, Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag, Berlin, pp. 91-103, 2001.
- [13] Huget, M. (2002) "**Agent UML Class Diagrams Revisited**," In Proceedings of Agent Technology and Software Engineering (AgeS), Bauer, B., Fischer, K., Muller, J. and Rumpe, B., Eds., Erfurt, Germany.
- [14] Huget, M. (2002) "**Generating Code for Agent UML Sequence Diagrams**." In Proceedings of Agent Technology and Software Engineering (AgeS), Bernhard Bauer, Klaus Fischer, Jorg Muller and Bernhard Rumpe (eds.), Erfurt, Germany, October.
- [15] Bauer, B. (2002) "**UML Class Diagrams revisited in the context of agent-based systems**." In: M. Wooldridge, P. Ciancarini, and G. Weiss (Eds.) Proceedings of Agent-Oriented Software Engineering, Second International Workshop, AOSE 2001, LNCS 2222 Springer, Canada, p. 101-118.
- [16] Huget, M-P., **An Application of Agent UML to Supply Chain Management**, in Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems, Bologna, Italy, 2002.
- [17] Beer, M., et al., **Designing Community Care Systems with AUML**, in Proceedings of the International Conference E.U-LAT e-Health, Cuernavaca, Mexico, 2003.
- [18] Odell, J., Parunak, H. and Fleisher, M., "**The Role of Roles in Designing Effective Agent Organizations**" In Software Engineering for Large-Scale Multi-Agent Systems, Garcia, A., Lucena, C., Zamobnelei, F., Omicini, A and Carstro, J., Eds., LNCS, Springer-Verlag, 2003.
- [19] Peres, J., Bergmann, U. **Experiencing AUML for MAS Modeling : A Critical View**, SEAS 2005, Uberlandia, MG.
- [20] FIPA Agent UML Web Site - <http://www.auml.org>, 2007.
- [21] FIPA ACL- Agent Communication Language Specification - <http://www.fipa.org/repository/aclspecs.html>, 2007.
- [22] Wooldridge, M., Jennings, N. and Kinny, D.; **The Gaia methodology for agent-oriented analysis and design**. Journal of Autonomous Agents and Multi-Agent Systems, 3(3):285-312, 2000.
- [23] Jennings, N. R.; **An agent-based approach for building complex software systems**. Communications ACM, 44(4):35-41, 2001.
- [24] Ciancarini, P. And Wooldridge, M., Agent-Oriented Software Engineering. Proceedings of the 1<sup>st</sup> International Workshop on Agent-Oriented Software Engineering, Springer Verlag, LNCS, Vol. 1957, pp. 1-24, 2001.
- [25] Zambonelli, F. and Omicini, A., 2004. **Challenges and Research Directions in Agent-Oriented Software Engineering**. Journal of Autonomous Agents and Multi-agent Systems, vol. 9, No. 3, Kluwer Academic Publishers, pp 253-283, 2004.

- [26] Boehm, B., **A Spiral Model of Software Development and Enhancement**. IEEE Computer, Vol. 21, N° 5, May, 1988, pp. 61-72, 1988.
- [27] Juan, T., Pearce, A. and Sterling, L., **ROADMAP: Extending the Gaia Methodology for Complex Open Systems**. Proceeding of the First International Conference on Autonomous Agents and Multi-Agent Systems - AAMAS '02, July 15-19, 2002, Bologna (Italy), pp. 3-10, 2002.
- [28] Padgham, L. and Winikoff, M., **Prometheus: A Methodology for Developing Intelligent Agents**. Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems - AAMAS '02, Third International Workshop on Agent-Oriented Software Engineering AOSE-2002, July 15, 2002, Bologna (Italy), pp. 135-146, 2002.
- [29] DeLoach, S., Wood, M. and Sparkman, C., **Multiagent Systems Engineering**. International Journal of Software Engineering and Knowledge Engineering, vol. 11, No. 3, pp. 231-258, 2001.
- [30] Wagner, G., **The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior**. Information Systems, Vol. 28, No. 5, July, 2003, Elsevier, pp. 475-504, 2003.
- [31] Lind, J., **Iterative Software Engineering for Multiagent Systems, the MASSIVE Method**. Springer Verlag, New York, Secaucus, NJ, USA, 2001.
- [32] Gómez-Sanz, J. and Pavón, J., **Agent Oriented Software Engineering with INGENIAS**. Proceedings of the 3rd Central and Eastern Europe Conference on Multi-agent Systems, Springer Verlag, LNCS 2691, pp. 394-403, 2003.
- [33] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J., **A Knowledge Level Software Engineering Methodology for Agent Oriented Programming**. In: Proceedings of the 5<sup>th</sup> International Conference on Autonomous Agents. ACM Press, Montreal (Canada), pp. 648-655, 2001.
- [34] d'Inverno, M., Fisher, M., Lomuscio, A., Luck, M., de Rijke, M., Ryan, M. and Wooldridge, M., **Formalisms for Multi-Agent Systems**. The Knowledge Engineering Review, 12(3): 315-321, 1997.
- [35] Balduino, R., Lyons, B., **OpenUP - A Process for Small and Agile Projects**, October 2006, [http://www.eclipse.org/epf/general/OpenUP\\_Basic.pdf](http://www.eclipse.org/epf/general/OpenUP_Basic.pdf) , accessed in July 2007.
- [36] FIPA. 2003. **Method fragment definition**. FIPA Document, <http://www.fipa.org/activities/methodology.html>.
- [37] Kruchten, P. 2000 **The Rational Unified Process: an Introduction, Second Edition**. 2nd. Addison-Wesley Longman Publishing Co., Inc.
- [38] Beck, K., **Test Driven Development**. Publisher Addison-Wesley, 240pg., 2002.
- [39] Coelho, R., Kulesza, U., von Staa, A., and Lucena, C., **Unit testing in multi-agent systems using mock agents and aspects**. In Proceedings of the 2006 interna-

tional Workshop on Software Engineering For Large-Scale Multi-Agent Systems (Shanghai, China, May 22 - 23, 2006). SELMAS '06. ACM Press, New York, NY, 83-90.

[40] Eclipse Process Framework Composer (EPF Composer), <http://www.eclipse.org/epf/>, July 2007.

[41] Bauer, B., Odell, J.: **UML 2.0 and Agents: How to Build Agent-based Systems with the new UML Standard**, Journal of Engineering Applications of Artificial Intelligence, Volume 18, Issue 2, March 2005, pp. 141-157.

[42] Luck, M., McBurney, P., Preist, Ch.: **Agent Technology: Enabling Next Generation Computing, A Roadmap for Agent Based Computing**. AgentLink, 2003, <http://www.agentlink.org/roadmap>.

[43] Fuggetta, A., 2000. Software Process: a Roadmap. Proceedings of the Conference on the Future of Software Engineering, June 4-11, 2000, Limerick (Ireland), ACM Press, New York (USA), pp. 25-34.

[44] Ghezzi, C., Jazayeri, M., and Mandrioli, D., 1991. Fundamentals of Software Engineering. Prentice Hall International, Upper Saddle River, NJ (USA).

[45] Beck, K. (2000) Extreme Programming Explained: Embrace Change, Addison-Wesley.

[46] Beck, K. (2001) Planning Extreme Programming, Addison-Wesley.

[47] Tiryaki, A. M., Öztuna, S., Dikenelli, O. and Erdur, R. C., Sunit: A unit testing framework for test driven development of multi-agent systems. In 7th International Workshop on Agent-Oriented Software Engineering, 2006. 3.5.