A Process Based on the Model-Driven Architecture to Enable the Definition of Platform-Independent Simulation Models

Alfredo Garro*, Francesco Parisi, and Wilma Russo

Department of Electronics, Computer and System Sciences (DEIS),
University of Calabria, via P. Bucci 41C, Rende (CS), 87036, Italy
{garro,w.russo}@unical.it, fparisi@deis.unical.it

Abstract. Agent-Based Modeling and Simulation (ABMS) offers many advantages for dealing with and understanding a great variety of complex systems and phenomena in several application domains (e.g. financial, economic, social, logistics, chemical, engineering) allowing to overcome the limitations of the classical and analytical modelling techniques. However, the definition of agent-oriented models and the use of the existing agent-based simulation platforms often require advanced modelling and programming skills, thus hindering a wider adoption of the ABMS mainly in those domains that would benefit more from it. To promote and ease the exploitation of ABMS, especially among domain experts, the paper proposes the jointly exploitation of both Platform-Independent Metamodels and Model-Driven approaches by defining a Model-Driven process (MDA4ABMS) which conforms to the OMG Model-Driven Architecture (MDA) and enables the definition of Platform-Independent simulation Models from which Platform-Dependent simulation Models and the related code can be automatically obtained with significant reduction of programming and implementation efforts.

Keywords: Agent-based Modeling and Simulation, Model-driven Development, Model-driven Architecture, Platform-independent Simulation Models.

1 Introduction

Approaches which combine agent-based modeling with simulation make it possible to support not only the definition of the model of a system at different levels of complexity through the use of autonomous, goal-driven and interacting entities (agents) organized into societies which exhibit emergent properties, but also the execution of the obtained model to simulate the behavior of the complete system so that knowledge of the behaviors of the entities (micro-level) produces an understanding of the overall outcome at the system-level (macro-level).

Despite the acknowledged potential of Agent-Based Modeling and Simulation (ABMS) for analyzing and modeling complex systems in a wide range of application

* Corresponding author.

domains (e.g. financial, economic, social, logistics, chemical, engineering) [30], these approaches are slow to be widely used as the obtained agent-based simulation models are at a too low-abstraction level, strongly platform-dependent, and therefore not easy to verify, modify and update [17, 22, 28]; moreover, significant implementation efforts which are even more for domain experts, typically lacking of advanced programming skills, are required [30]. In particular, agent-based simulation models can be currently obtained mainly through either direct implementation or manual adaption of a conceptual system model for a specific ABMS platform. The former approach inevitably suffers from the limitations and specific features of the chosen platform, whereas the latter requires additional adaptation efforts, the magnitude of which increases depending on the gap between the conceptual and implementation models of the system.

To overcome these issues, solutions based on approaches well-established in contexts other than the ABMS can be exploited; in particular: (i) approaches based on Platform-Independent Metamodels, which enable the exploitation of more high-level design abstractions in the definition of Platform-Independent Models and the subsequent automatic code generation for different target platforms [1]; (ii) Model-Driven approaches, which enable the definition of a development process as a chain of model transformations [4]. Therefore, some solutions for the ABMS context currently exploit either the approach based on Platform-Independent Metamodels [3, 21, 30] or that based on Model-Driven [18, 22, 28]. The former approach makes available in this context the benefits of exploiting the high level abstraction typical of Platform-Independent Models which also enables the exchange of models regardless of the specific platform used for the simulation; in addition, Platform-Independent Models can be reviewed by domain experts working on different target platforms (possibly on the basis of the simulation result obtained), and then shared with other domain experts. The latter approach enables the definition of complete and integrated processes able to guide domain experts from the analysis of the system under consideration to its agent-based modeling and simulation. In fact, according to the Model-Driven paradigm, the phases which compose a process, the work-products of each phase and the transitions among the phases in terms of model transformations are fully specified; in addition, as the Model-Driven paradigm makes it possible the automatic code generation from a set of (visual) models of the system, the focus can be geared to system modeling and simulation analysis rather than to programming and implementation issues.

Under these considerations, this paper proposes the jointly exploitation of both Platform-Independent Metamodels and Model-Driven approaches as a viable solution able to fully address the highlighted issues so to promote a wider adoption of the ABMS especially in those domains that would benefit more from it. In particular, the paper proposes a Model-Driven process [4] able to guide and support ABMS practitioners in the definition of Platform-Independent Models starting from a conceptual and domain-expert-oriented modeling of the system without taking into account simulation configuration details. The proposed process conforms to the OMG Model-Driven Architecture (MDA) [32] and then allows to (automatically) produce *Platform-Specific simulation Models* (PSMs) starting from a *Platform-Independent simulation Model* (PIM) obtained on the basis of a preliminary *Computation Independent Model* (CIM).

The remainder of this paper is organized as follows: available ABMS languages, methodologies and tools are briefly discussed along with the main drawbacks which still hinder their wider adoption (Section 2), the proposed MDA-based process for ABMS (MDA4ABMS) is presented (Section 3) and then exemplified (Section 4) with reference to a popular problem (the *Demographic Prisoner's Dilemma*) able to represent several social and economic scenarios. Finally, conclusions are drawn and future works delineated.

2 ABMS: Languages, Methodologies and Tools

Several approaches have been proposed to support the definition of agent-based models and/or their implementation for specific simulation platforms; in the following, these approaches, grouped on the basis of the main features they provide, are briefly discussed and their main drawbacks, which still hinder their wider adoption, are highlighted.

1. *Agent-based Modeling and Simulation Platforms.* ABMS platforms, which also provide a visual editor for defining simulation models and, in particular, for specifying agent behaviours, as well as semi-automatic code generation capabilities, are currently available, e.g. Repast for Python Scripting (RepastPy) [11], Repast Simphony (Repast S) [29], the Multi-Agent Simulation Suite (MASS) [19], Ascape [35], SeSAm [25], and Escape [3].

Although the existing ABMS tools attempt to offer *comfortable* modeling and simulation environments, their exploitation is *comfortable* only when used for simple models. In fact, to model complex systems where basic behavior templates provided by the tools must be extended, significant programming skills are essential. Moreover, as these tools do not refer to any specific ABMS process, their use is mainly based on the extension and refinement of the examples and case studies provided, thus limiting such platform-dependent models to lower levels of abstraction and flexibility. Finally, the agent models adopted are often purely reactive and do not take into account organizational issues.

2. *Agent-based Modeling Languages.* Agent modeling languages, mainly coming from the Agent-Oriented Software Engineering (AOSE) domain, can be exploited for a clear, high level and often semantically well-founded definition of ABMS models; some of the wider adopted proposals are the Agent-Object-Relationship (AOR) Modeling [42], the Agent UML (AUML) [5], the Agent Modeling Language (AML) [10] and the Multi-Agent Modelling Language (MAML) [20].

These languages, which do not refer to a specific modeling process, are high-level languages based on graphical and, in some cases, easily customizable notations. Their capabilities make them more suitable as languages for depicting models than as programming languages. Moreover, compared to the models offered by agent-based simulation toolkits, the agent models expressed by these languages are richer, both at micro (agent) and macro (organization) levels. However, the definition of these agent models often requires advanced modeling skills and the transition from the produced design models to specific operational models must be often manually performed; this task, in absence of tools enabling (semi)automatic transitions, can be quite difficult due to the consistent gap between the design and the operational model of the system.

3. *AOSE Processes and Methods for Agent-based Modeling and Simulation.* Processes and methodologies for the analysis, design and implementation of agent-based models, can be derived from the AOSE domain and possibly adapted for ABMS. Specifically, among the several available AOSE methodologies (such as PASSI [13], PASSIM [14], ADELFE [7], GAIA [43] and GAIA2 [44], TROPOS [8], SONIA [2], SODA+zoom [27], MESSAGE [9], INGENIAS [36], O-MaSE [16], SADDE [39], and Prometheus [34]), some of these, such as GAIA2 [44], SODA+zoom [27] and MESSAGE [9], provide processes, techniques and/or abstractions which are particularly suited for the ABMS context; moreover, specific ABMS extensions of AOSE methodologies can be found in [23, 37, 40].

Although these proposals can represent reference methods for guiding domain experts through the different phases of an ABMS process, only few of them go beyond the high level design phase and deal with detailed design and model implementation issues. As a consequence, they fail in supporting domain experts in the definition of agent-based models which can be directly and effortless executed on ABMS platforms able to fully handle the phases of simulation and result analysis. In fact, the adaptation between the models obtained and the target simulation models requires significant efforts which are time-consuming, error-prone and demands advanced programming skills.

4. *Model-driven Approaches for ABMS.* To fully support and address not only the design but also the implementation of simulation models on available ABMS platforms, some Model-Driven approaches for ABMS have been proposed [18, 22, 28]. However, as they refer to specific ABMS platforms, their exploitation is strongly related to the adoption of these platforms (e.g. Repast Simphony for [18], BoxedEconomy for [22], MASON for [28]).

With reference to other MDA-based approaches, which aim to provide a methodological support for the design of agent-based distributed simulations compliant to the High Level Architecture (HLA) [12, 41], in the ABMS context a still debated issue [26] concerns the trade-off between the overhead which the HLA layer introduces and the provided distribution and interoperability benefits. Specifically, some approaches conceive HLA as the PSM level of an MDA Architecture and provide a process for transforming a System PIM, based on UML, in a HLA-based System PSM [12]. On the contrary, HLA is conceived as the PIM level in [41] where the Federation Architecture Metamodel (FAMM) for describing the architecture of a HLA compliant federation is proposed to enable the definition of platform-independent simulation models (based on HLA) and the subsequent code generation phase.

3 The MDA4ABMS Process

This section describes the proposed MDA4ABMS process which combines the Model-Driven approach and the exploitation of Platform-Independent Metamodels so making available in the ABMS context the benefits of both exploited approaches. The MDA4ABMS process relies on the Model-Driven Architecture (MDA) [32] and the

Agent Modeling Framework (AMF) which is proposed in [3] for supporting the platform-independent modeling.

MDA, which is the most mature Model-Driven proposal launched by the Object Management Group (OMG), is based on the process depicted in Figure 1 where three main abstraction levels of a system and the resulting model transformations are introduced; in particular, the models related to the provided abstraction levels are the following:

- a *Computation Independent Model* (CIM) which describes context, requirements and organization of a system at a conceptual level;
- a *Platform-Independent Model* (PIM) which specifies architectural and behavioral aspects of the system without referring to any specific software platform;
- the *Platform-Specific Models* (PSMs) which describe the realization of the system for specific software platforms and from which code and other development artifacts can be straightforwardly derived.

Transformations between these models (M1 Layer) are enabled by both the corresponding *metamodels* in the M2 Layer and the mappings among metamodels. Each metamodel is defined as instance of the *meta-metamodel* represented in the M3 Layer by the Meta Object Facility (MOF) [31].

The MDA process provides the reference architecture for supporting the generation of target models given a source model as well as the mapping between its metamodel and the target metamodels. To exploit the MDA process in the ABMS domain and obtain agent-based models for specific platforms starting from a platform-independent model, the basic MDA concepts, which have been specifically conceived for the Software Engineering domain, have to be mapped into the ABMS counterparts.

To address these issues, the proposed MDA4ABMS process characterizes the following items (which are highlighted in Figure 1): (i) a reference CIM metamodel for the definition of CIMs which supports the agent-based conceptual system modeling carried out through both abstract and domain-expert oriented concepts (see Section 3.1); (ii) a PIM metamodel for the definition of Platform-Independent ABMS Models (See Section 3.2); (iii) mappings among these metamodels so to enable ABMS model transformations (see Section 3.3). The solution identified for the PIM level allows the

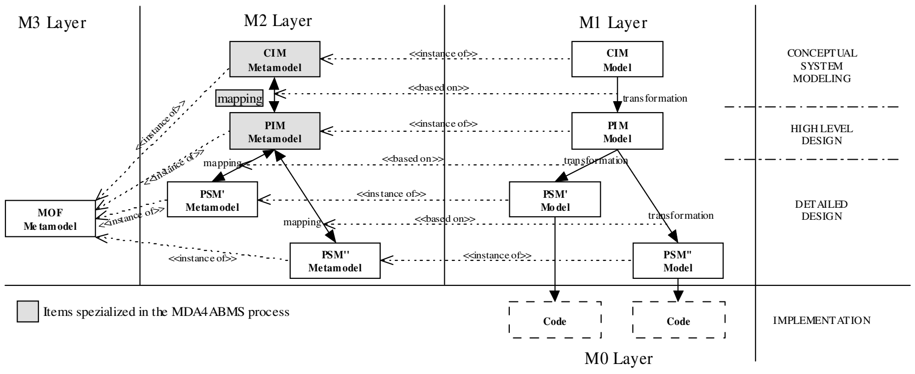


Fig. 1. The MDA-based process

automatic generation of PSMs and the related code for several popular ABMS platforms [3, 29, 35]; on the basis of the provided PIM metamodel, other PSM metamodels for the definition of PSM models can be defined for other and new simulation platforms.

3.1 The CIM Metamodel

The CIM metamodel of the MDA4ABMS process is defined by adopting for the behavior of agents a light and task-based model which combines the strengths of several well-known, task-based agent models [6]. This metamodel is quite general and plain, as required by the abstraction level for which it has been conceived, but powerful enough for representing, at a conceptual level, a great variety of systems in typical ABMS domains.

In particular, the CIM metamodel reported in Figure 2 is centered on the concept of *Agent*. An *Agent*, which is situated in an *Environment* constituted by *Resources,* is characterized by a *Behavior* and a set of *Properties*. *Agents* can be organized into *Societies* which in turn can be organized in *sub-societies*. A *Behavior* is composed by a set of *Tasks* organized according to *Composition Task Rules* which define precedence relations between *Tasks*. Each *Task*, which can act on a set of environment's *Resources*, is structured as an UML 2.0 Activity Diagram which consists of a set of linked *Actions* that can be either *Control Flow* (pseudo) actions (i.e. *start*, *end*, *split*, *join*, *decision*, *merge*, *sequence*) or *Computation* and *Interaction* actions (i.e. *outgoing* or *incoming signals*).

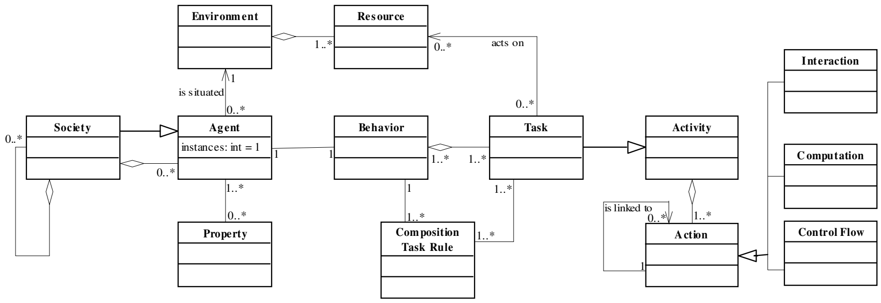


Fig. 2. The CIM metamodel

3.2 The PIM Metamodel

The definition of a PIM metamodel, able to represent a reference metamodel for the definition of Platform-Independent ABMS Models from which different Platform Specific Models (PSMs) can be derived, results in a challenging, long-term standardization process which should also take into account the features of the main ABMS platforms. A more practical solution can be based on the exploitation of the Agent Modeling Framework (AMF) [3] which is meant to provide a reference representation of platform-independent models that can be used to generate simulation models for widely adopted ABMS platforms. In particular, by using the AMF approach, PIM models can be defined through a hierarchical visual editor and represented by XML documents [38] which are exploited for the generation of PSMs and related code.

Starting from the AMF proposal, the PIM metamodel of the MDA4ABMS process (see Figure 3) has been effortlessly defined. This metamodel is centered on the concept of (Simulation) *Context* (*SContext*) which represents an abstract environment in which (Simulation) *Agents* (*SAgents*) can act. An *SAgent* is provided with an internal *state* consisting of a set of *SAttributes*, a visualization style *SStyle*, and a group of *AActs* (*AGroup*) which constitute its behavior. An *AAct* is characterized by an *Execution Setting* which establishes when its execution can start, its periodicity and its priority.

SContexts, which are themselves *SAgents*, can be organized hierarchically and contain *sub-SContexts*. *SAgents* in an *SContext* can be organized by using *SProjections* which are structures designed to define and enforce relationships among *SAgents* in the *SContext*. In particular, a *SNetwork* projection defines the relationships of both acquaintance and influence between *SAgents* whereas *SGrid, SSpace, SGeography* and *SValueLayer* projections define either the physical space or logical structures in which the agents can be situated.

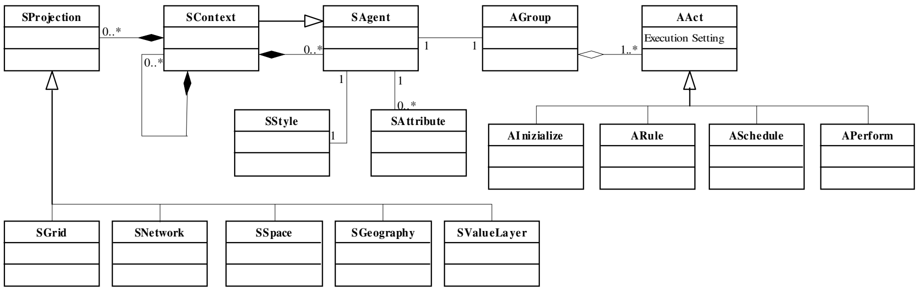


Fig. 3. The PIM metamodel

3.3 From CIM to PIM

With reference to an MDA-based process, a target model can be obtained by transforming a source model (M1 Layer in Figure 1) on the basis of the mapping between the source and target metamodels (M2 Layer in Figure 1). To this end, to enable the definition of instances of concepts of the target metamodel from instances of concepts of the source metamodel, mapping rules among the corresponding concepts along with additional guidelines should be provided [24, 32].

This section, which deals with the mapping between the CIM and PIM metamodels (see Section 3.1 and 3.2) of the MDA4ABMS process, provides the mapping rules (Section 3.3.2) and some guidelines (Section 3.3.3) enabling to transform the CIM entities into PIM entities by taking into account specific aspects (see Section 3.3.1) of the AMF-based PIM metamodel. The subsequent generation of several PSMs (and code for the related ABMS platforms) from the obtained PIM can be then easily carried out by the visual and Eclipse-based modelling environment provided by the AMF framework [3].

3.3.1 Main Aspects of an AMF-Based PIM

Some main aspects have to be considered in the definition of an AMF-based PIM; in this section, the focus is on those which are relevant since they affect the simulation execution of the derived PSMs and which, in particular, concern the proper definition of the *Execution Setting* of an AAct, and the exploitation of *SAttributes* to enable communication among SAgents (see Figure 3).

An AMF-based PIM is defined according to a time-stepped driven simulation approach (the simulation time is incremented in fixed steps) [30], in which, at each simulation step *t*, a set of AAct instances which can be executed and their execution order are defined. Specifically, in a step *t:* (i) for each AAct, belonging to the *AGroup* of an *SAgent* SA, the number of its instances depends on the number of SA instances; (ii) the AAct *Execution Settings* determine the AAct instances to be executed and their execution order.

The Execution Setting of an AAct is characterized by the tuple <*startingTime, period, priority*> where:

- *startingTime* is the first simulation step at which the instances of the AAct are to be executed;
- for each instance of the AAct, *period* is the number of simulation steps which must elapse between two subsequent executions;
- in a simulation step the *priority* value affects the execution order of the *enabled* AActs instances (an AAct is *enabled* at the *simulation step t* if *t* is equal to the AAct *startingTime* which is incremented by a multiple of its *period*).

In a simulation step *t* all enabled AAct instances (regardless of whether they belong to a specific SAgent instance) belong to the same set, *Enabled(t)*, from which the AActs are scheduled for execution on the basis of their *priority* (see Figure 4). As a consequence, the AAct Execution Settings have to be properly defined to guarantee right execution order between AAct instances of both the same SAgent instance (intra-agent AAct interleaving) and different SAgent instances (inter-agent AAct interleaving).

Moreover, in defining the AAct Execution Settings, the different AAct types should be also considered (see Figure 3). In particular, *AActs* of type *AInitialize* are executed once and before any other *AAct* of the *SAgent* (*starting Time* and *period* are both fixed to 0), *AActs* of type *ARule* are executed once at each iteration (*starting Time* and *period* are both fixed to 1), no fixed settings are associated to *AActs* of the *ASchedule* and *APerform* types as *ASchedule* supports periodicity greater than that of

```
ActScheduling (t) {
  AAI = Enabled(t); /* Enabled(t) returns the set of enabled AAct instances at t */
  while (not empty AAI) {
      MPE = maxPriorityEnabled(AAI) ; /* maxPriorityEnabled(AAI) returns a set
                            consisting of the AAct instances with maximum priority in AAI */
      AAI = AAI - MPE;
      while (not empty MPE) {
          aa = randomGet(MPE); /* randomGet(MPE) returns an AAct instance randomly
                                     chosen in (and removed from) MPE */
          execute (aa);
      }
  }
}
```

Fig. 4. Execution of an AMF-based simulation step

a single iteration, whereas *AActs* of type *APerform*, in each iteration in which they are scheduled, are over and over again executed until their escape conditions are met.

With respect to the communication among SAgents, since the *SAttributes* of an SAgent can be freely accessed by all the instances of the SAgent, and the SAttributes of an SContext by all the instances of all the SAgents in the SContext, communication among instances of the same SAgent (intra-agent communication) can exploit SAgent *SAttributes* whereas communication among instances of different SAgents (inter-agent communication) can be enabled by SContext *SAttributes*.

Finally, the design of SAgent communications should take into account how random choices among the enabled AAct (see Figure 4) affect the values of the *SAttributes* on which the communication is based.

3.3.2 Mapping from CIM to PIM Metamodels: Mapping Rules

The automatic generation of a PIM starting from a given CIM is enabled by the QVT/R-based representation of mapping rules [32, 33]. Specifically, due to the different abstraction level between the concepts of the reference CIM and PIM metamodels (see Section 3.1 and 3.2), the mapping rules introduced in this Section along with the QVT/R-based representation allow to obtain a preliminary PIM which needs to be refined by applying additional guidelines (see Section 3.3.3).

The preliminary transformation of a CIM into a PIM, which involves the definition of instances of concepts of the PIM metamodel from instances of concepts of the CIM metamodel by exploiting the mapping rules among the corresponding concepts, consists in the following steps which are listed in the order they should be performed:

R1. each *Society* is transformed into a Simulation Context (*SContext*) and any enclosed Society into a (sub)*SContext* of the corresponding enclosing *Society*; *SAttributes* of each *SContext* are, then, originated by the *Properties* of the corresponding *Society*;

R2. each *Agent* belonging to a *Society* is transformed into an *SAgent* of the corresponding SContext, generating the *SAgent SAttributes* on the basis of the *Agent Properties*, and introducing the *SAgent AGroup* which groups the *AActs* constituting its behavior;

R3. on the basis of the set of *Resources*, which compose the *Environment* in which *Agents* are situated, a set of *SProjections*, whose types (SNetwork, SGrid, SSpace, SGeography, SValueLayer) depend on the characteristics of the mapped *Resources*, are then introduced in the corresponding *SContext*;

R4. *AActs* associated to each *SAgent* are to be defined on the basis of the behavior of the corresponding *Agent* which is composed by a set of *Tasks* organized according to *Composition Task Rules*; this transformation is not direct as requires to take into account the specific aspects of both an AMF-based PIM (see Section 3.3.1) and the simulation scenarios to be represented;

R5. *Actions* which constituted the *Tasks* mapped into an *AAct* have to be properly realized by exploiting the wide set of predefined functions provided by AMF [3].

With reference to the above introduced rules, in the following, some guidelines are provided which address some relevant issues related to: (i) the different communication mechanisms adopted by CIM and PIM metamodels, the former based on incoming and outgoing signals (see Section 3.1), and the latter on shared *SAttributes* (see Section 3.3.1); (ii) the setting of both *AAct Execution Settings* and related *AAct types* which have to ensure compliance with the *Composition Rules* of the corresponding *Tasks*. Moreover, *AAct Execution Settings* and related *AAct types* should also be set to guarantee intra and inter-agent *AAct* interleavings (see Section 3.3.1) which adhere to the simulation scenarios under consideration.

The QVT/R-based representation of the above introduced mapping rules is exemplified in Figure 5 where the rule R2 for transforming an Agent into an *SAgent* is reported by using the QVT/R graphical notation [33].

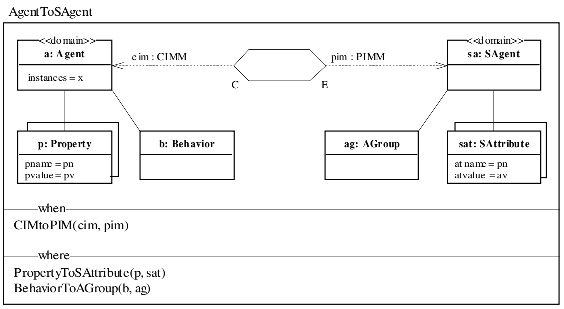


Fig. 5. The QVT/R graphical notation: rule R2

3.3.3 Mapping from CIM to PIM Metamodels: Guidelines

Beside the above introduced *mapping rules* among concepts of the source and target metamodels, further support for CIM to PIM transformation can be provided through *guidelines* which take into account not only the different abstraction level of the concepts in the metamodels but also the main aspects related to the simulation execution model of an AMF-based PIM (see Section 3.3.1). In particular, these *guidelines* propose viable solutions for guiding the choice among the mapping alternatives which often characterize the transformation process from a conceptual level (CIM) to a less abstract level (PIM) typically relying on a simulation execution model. In the following some of these *guidelines* are proposed and exploited in Section 4:

G1. A set of *Tasks* of an *Agent* which, according to the *Composition Task Rules*, can be grouped in a sequence of *Tasks* and in which *Tasks* are related by *Actions* of the *Interaction* type (i.e. the involved *Tasks* send/receive messages to/from the other *Tasks* in the sequence) can be mapped in a single *AAct* of an *SAgent*. The interactions among the involved *Tasks* are then modeled by accessing and modifying the properly introduced *SAttributes* of the *SAgent*.

G2. In case of *Tasks* which should be executed at the same simulation steps, the *Execution Setting* of the resulting *AActs* must have the same *startingTime* and *period* whereas *priorities* must be properly set according to the task organization specified by the *Composition Task Rules*.

G3. The *SAttributes* of an *SContext* should be properly defined not only for mapping the *Properties* of the corresponding *Society* but also for supporting interactions among different *SAgents* belonging to the *SContext*.

G4. *Tasks* (or group of *Tasks*) that must be executed at every simulation step are mapped into *ARules*, except for *Tasks* containing a *Do-While loop* which should be mapped into *APerforms*. Tasks that must be executed with a periodicity different from a single simulation step should be mapped into *ASchedule*; finally, *Tasks* that must be executed once before any other *Tasks* should be mapped into *AInizialize* (an *AAct* of type *AInitialize* should nevertheless be provided for setting the *SAttributes*).

4 Exploiting the Proposed MDA-Based Process

In this section, the MDA4ABMS process is exemplified with reference to the well-known *Demographic Prisoner's Dilemma* which was introduced by Epstein in 1998 [15] and is able to represent several social and economic complex scenarios in which interesting issues regard the identification of starting configurations and conditions that allow initial populations to reach stable configurations (in terms of both density and geographic distribution). Specifically, in these scenarios k players are spatially distributed over an n-dimensional toroidal grid. Each player is able to move to empty cells in its von Neumann neighborhood of range 1 (*feasible* cells), is characterized by a fixed pure *strategy* (c for cooperate or d for defect) and is endowed with a level of wealth w which will be decremented or incremented depending of the payoff earned by the player in each round of the *Prisoner's Dilemma* game played during its life against its neighbors [15]. The player dies when its wealth level w becomes negative, whereas, when w exceeds a threshold level w_b, an *offspring* can be produced with wealth level w_0 deducted from the parent and plays using the same strategy as the parent unless a mutation (with a given rate m) occurs. A player also dies if its *age* exceeds a value age_{max} randomly fixed at the player creation.

4.1 The CIM Model

For the *Demographic Prisoner's Dilemma*, the CIM model envisages a *DPDGame* Society of k *Player* Agents which are situated in an Environment which includes a *Grid* Resource constituted by an n-dimensional toroidal grid. Main Properties of the *DPDGame* Society are Prisoner's Dilemma *payoffs*, initial and threshold wealth levels (w_0, w_b), and mutation rate (m), and those of the *Player* Agent are its wealth level w, *age*, and *strategy*. The Behavior of the *Player* Agent is obtained by composing the set of Tasks reported in Table 1 according to the Composition Task Rules shown in Table 2; corresponding UML Activity diagrams are reported in Figure 6.

4.2 The PIM Model

In this section, the transformation from the defined CIM to a PIM is detailed with reference to a simulation scenario where all players are required to play exactly one round in a simulation step.

The transformation from the CIM to a PIM is enabled by the mapping between the CIM and PIM metamodels (see Sections 3.3.2, 3.3.3) which originates: the *DPDGame* SContex from the DPDGame Society (*rule R1*), the *Player* SAgent from the Player Agent (*rule R2*), the *GameSpace* SProjection from the *Grid* Resource (*rule R3*), the *Acts* (with their related *Execution Settings*) associated to the *Player* SAgent from the Tasks and associated Composition Task Rules composing the Behavior of the Player (*rule R4*).

Table 1. Indentified tasks

Task Id	Task Name	Description
T1	Walk	The player can move to a *feasible* cell of the *Grid*.
T2	Challenge	*If the von Neumann neighborhood (of range 1) of the player is not empty* the player communicates its strategy to its randomly selected opponent player.
T3	Update Age	The player age is incremented by 1.
T4	Fission	*If the player's wealth level w is greater than the threshold w_b* a new child player can be created in a *feasible* cell of its parent and endowed with w_0 and the same strategy of the parent (unless a mutation with rate *m* occurs). The wealth level of the parent player is decremented by w_0.
T5	Die	*If the wealth level of the player is negative or its age is greater than age_{max}* the player is removed from the *Grid*.
T6	Accept Dare	*When the strategy of an opponent player is provided* the player strategy is communicated to the opponent and the earned payoff is added to the player's wealth level.
T7	Update Wealth Level	*If the strategy of an opponent player is provided* the earned payoff is added to the player's wealth level

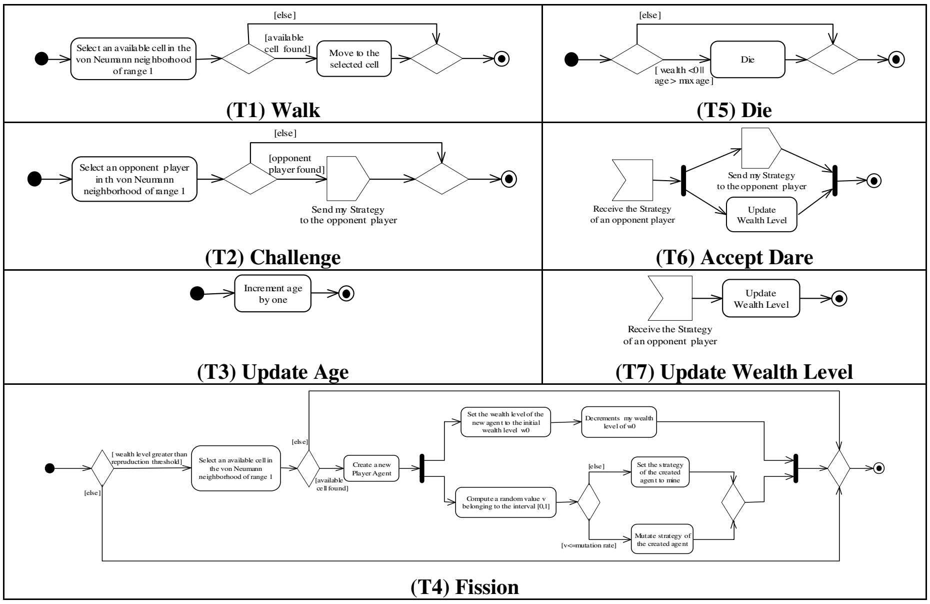

[else]
Select an available cell in the von Neumann neighborhood of range 1
[available cell found]
Move to the selected cell
(T1) Walk
[else]
Die
[wealth <0 || age > max age]
(T5) Die
[else]
Select an opponent player in th von Neumann neighborhood of range 1
[opponent player found]
Send my Strategy to the opponent player
(T2) Challenge
Receive the Strategy of an opponent player
Send my Strategy to the opponent player
Update Wealth Level
(T6) Accept Dare
Increment age by one
(T3) Update Age
Receive the Strategy of an opponent player
Update Wealth Level
(T7) Update Wealth Level
[wealth level greater than repruduction threshold]
[else]
Select an available cell in the von Neumann neighborhood of range 1
[else]
[available cell found]
Create a new Player Agent
Set the wealth level of the new agent to the initial wealth level w0
Decrements my wealth level of w0
Compute a random value v belonging to the interval [0,1]
[else]
Set the strategy of the created agent to mine
[v<=mutation rate]
Mutate strategy of the created agent
(T4) Fission

Fig. 6. The UML activity diagrams of the Player Agent tasks

Table 2. Composition task rules

Task Id	Set of Enabling Tasks
T1	-
T2	{T1}
T3	{T1}
T4	{T7}
T5	{T3, T4}
T6	{T2}
T7	{T6}

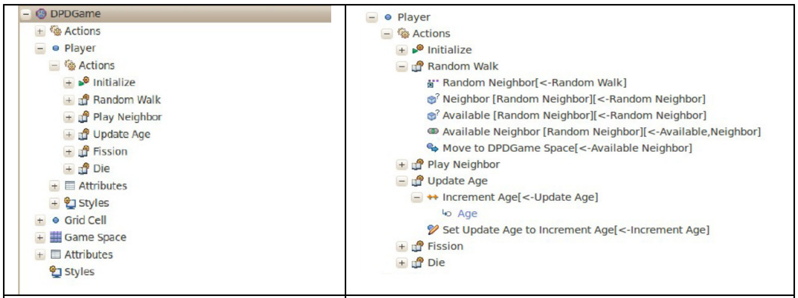
Table 3. Group of acts (AGroup) for the player agent

AAct	**AAct Execution Setting**	**Tasks**
Random Walk	<1,1, a>	T1
Play Neighbor	<1,1, b>, with b<a	T2, T6, T7
Update Age	<1,1, c>, with c<a	T3
Fission	<1,1, d>, with d<c & d<b	T4
Die	<1,1, e>, with e<d	T5

In Table 3 the Acts derived for the *Player* SAgent along with the associated Tasks (see Table 1 and 2) and Execution Settings are reported. As the AMF communication mechanism among instances of an SAgent is based on access to the SAttributes of the SAgent (see Section 3.3.1), a single AAct (*Play Neighbor*) is derived from tasks T2, T6 and T7 which carried out this kind of communication (*guideline G1*). Execution Settings of the AActs in Table 3 are characterized by both *startingTime* and *period* equal to one to guarantee that all the Player SAgents perform all their AActs in each simulation step, and *priorities* are set (*guideline G2*) on the basis of the Compositions Task Rules (see Table 2). On the basis of the AAct *Execution Setting* (see Section 3.3.1) in Table 3 the type of AActs is obtained (*guideline G4*).

In Figure 7.a an example of a PIM model representation, obtained by exploiting the visual and Eclipse-based modelling environment provide by AMF, is reported. Moreover, an AAct of the AInizialize type (*Inizialize*) has been introduced for setting up the *SAttributes* of the *DPDGame* SContext and the *Player* SAgent *(guideline G4)*. In Figure 7.b. the definition of the *Random Walk* and *Update Age* AActs is reported where the actions associated to each AAct are defined by exploiting the wide set of functions provided by AMF (*rule R5*).

Starting from this definition of the PIM model, AMF is able to automatically generate the PSM models and the related code for the ABMS platforms which are currently supported: Repast Simphony [29], Ascape [35] and Escape [3]. The simulation of the system can then be executed in a target simulation environment and simulation results can be thoroughly analyzed by exploiting several analysis tools (as Matlab, R, VisAd, iReport, Jung) which can be directly invoked from the environment.



```
− DPDGame
  + Actions
  − Player
    − Actions
      + Initialize
      + Random Walk
      + Play Neighbor
      + Update Age
      + Fission
      + Die
    + Attributes
    + Styles
  + Grid Cell
  + Game Space
  + Attributes
    Styles
```

```
− Player
  − Actions
    + Initialize
    − Random Walk
        Random Neighbor[<-Random Walk]
        Neighbor [Random Neighbor][<-Random Neighbor]
        Available [Random Neighbor][<-Random Neighbor]
        Available Neighbor [Random Neighbor][<-Available,Neighbor]
        Move to DPDGame Space[<-Available Neighbor]
    + Play Neighbor
    − Update Age
      − Increment Age[<-Update Age]
          Age
        Set Update Age to Increment Age[<-Increment Age]
    + Fission
    + Die
```

(a) *DPDGame* model in AMF (b) *Random Walk* and *Update Age* AActs

Fig. 7. The AMF-based PIM model of the DPDGame

5 Conclusions

A wider adoption of the ABMS is still hindered by the lack of approaches able to fully support the experts of typical ABMS domains (e.g. financial, economic, social, logistics, chemical, engineering) in the definition and implementation of agent-based simulation models. In this context, the paper has proposed a solution, centered on the joint use of the Model-Driven Architecture and AMF-based Platform-Independent Metamodel, which aims to overcome the main drawbacks of available ABMS languages, methodologies and tools. In particular, the proposed process (MDA4ABMS) allows to (automatically) produce Platform-Specific simulation Models (PSMs) starting from a Platform-Independent simulation Model (PIM) obtained on the basis of a Computation Independent Model (CIM), thus allowing domain experts to exploit more high-level design abstractions in the definition of simulation models and to exchange/update/refine the so obtained simulation models regardless to the target platform chosen for the simulation and result analysis. Moreover, the semi-automatic model transformations, enabled by the defined metamodels and related mappings, ease the exploitation of the proposed modeling notation and process, while the adoption of the standard UML notation and the visual modeling tool provided by AMP reduce the learning curve of the process.

The MDA4ABMS process has been exemplified with reference to the well-known *Demographic Prisoner's Dilemma* which is able to represent several social and economic complex scenarios thus demonstrating the efficacy of the process and the related tools in supporting domain experts from the definition of conceptual simulation models to their concrete implementation on different target ABMS platforms.

Ongoing research efforts are devoted to: (i) define and extensive experiment a full-fledged ABMS methodology based on the MDA4ABMS process and able to seamlessly guide domain experts from the analysis of a complex system to its agent-based modeling and simulation; (ii) look for frameworks different from AMF (e.g. HLA) suitable to define PIM metamodels able to support the modeling of simulation scenarios with specific requirements such as distribution and/or human participation.

References

1. Agt, H., Bauhoff, G., Cartsburg, M., Kumpe, D., Kutsche, R., Milanovic, N.: Metamodeling Foundation for Software and Data Integration. In: Yang, J., Ginige, A., Mayr, H.C., Kutsche, R.-D. (eds.) UNISCON 2009. LNBIP, vol. 20, pp. 328–339. Springer, Heidelberg (2009)
2. Alonso, F., Frutos, S., Martínez, L., Montes, C.: SONIA: A Methodology for Natural Agent Development. In: Gleizes, M.-P., Omicini, A., Zambonelli, F. (eds.) ESAW 2004. LNCS (LNAI), vol. 3451, pp. 245–260. Springer, Heidelberg (2005)
3. The AMP project, `http://www.eclipse.org/amp/`
4. Atkinson, C., Kühne, T.: Model-driven development: A metamodeling foundation. IEEE Software 20(5), 36–41 (2003)
5. Bauer, B., Müller, J.P., Odell, J.: Agent UML: A Formalism for Specifying Multiagent Software Systems. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957, pp. 91–103. Springer, Heidelberg (2001)

6. Bernon, C., Cossentino, M., Gleizes, M.-P., Turci, P., Zambonelli, F.: A Study of Some Multi-agent Meta-models. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382, pp. 62–77. Springer, Heidelberg (2005)
7. Bernon., C., Gleizes, M.P., Picard, G., Glize, P.: The Adelfe Methodology for an Intranet System Design. In: Proc. of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS), Toronto, Canada (2002)
8. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: TROPOS: an agent-oriented software development methodology. Journal of Autonomous Agents and Multi-agent Systems 8(3), 203–236 (2004)
9. Caire, G., Coulier, W., Garijo, F.J., Gomez, J., Pavón, J., Leal, F., Chainho, P., Kearney, P.E., Stark, J., Evans, R., Massonet, P.: Agent Oriented Analysis Using Message/UML. In: Wooldridge, M.J., Weiß, G., Ciancarini, P. (eds.) AOSE 2001. LNCS, vol. 2222, pp. 119–135. Springer, Heidelberg (2002)
10. Cervenka, R., Trencansky, I.: The Agent Modeling Language - AML. Whitestein Series in Software Agent Technology. Birkhäuser (2007)
11. Collier, N., North, M.: Repast for Python Scripting. In: Proc. of the Agent 2004 Conference on Social Dynamics: Interaction, Reflexivity and Emergence, Chicago, IL (2004)
12. D'Ambrogio, A., Iazeolla, G., Pieroni, A., Gianni, D.: A Model Transformation approach for the development of HLA-based distributed simulation systems. In: Proc. of the International Conference on Simulation and Modeling Methodologies, Technologies and Applications, Noordwikerhout, The Netherlands, July 29-31 (2011)
13. Cossentino, M.: From requirements to code with the PASSI methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, pp. 79–106. Idea Group Inc., Hershey (2005)
14. Cossentino, M., Fortino, G., Garro, A., Mascillaro, S., Russo, W.: PASSIM: a simulation-based process for the development of Multi-Agent Systems. J. of Agent-Oriented Software Engineering 2(2), 132–170 (2008)
15. Dorofeenko, V., Shorish, J.: Dynamical Modeling of the Demographic Prisoner's Dilemma. In: Computing in Economics and Finance. Society for Computational Economics (2002)
16. Garcia-Ojeda, J.C., DeLoach, S.A., Robby, R., Oyenan, W. H., Valenzuela, J.: O-MaSE: A Customizable Approach to Developing Multiagent Development Processes. In: Proc. of the 8th International Workshop on Agent Oriented Software Engineering, Honolulu HI (May 2007)
17. Garro, A., Russo, W.: Exploiting the easyABMS methodology in the logistics domain. In: Proceedings of the Int'l Workshop on Multi-Agent Systems and Simulation (MAS&S 2009) as Part of the Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW 2009), Turin, Italy, September 7-11 (2009)
18. Garro, A., Russo, W.: easyABMS: a domain-expert oriented methodology for Agent Based Modeling and Simulation. Simulation Modeling Practise and Theory 18, 1453–1467 (2010)
19. Gulyás, L., Bartha, S., Kozsik, T., Szalai, R., Korompai, A., Tatai, G.: The Multi-Agent Simulation Suite (MASS) and the Functional Agent-Based Language of Simulation (FABLES). In: SwarmFest 2005, Torino, Italy, June 5-7 (2005)
20. Gulyas, L., Kozsik, T., Corliss, J.B.: The multi-agent modelling language and the model design interface. J. of Artificial Societies and Social Simulation 2(3) (1999)
21. Hahn, C., Madrigal-Mora, C., Fischer, K.: Interoperability through a Platform-Independent Model for Agents. In: Enterprise Interoperability II, New Challenges and Approaches. Springer (2007)

22. Iba, T., Matsuzawa, Y., Aoyama, N.: From Conceptual Models to Simulation Models: Model Driven Development of Agent-Based Simulations. In: Proc. of the 9th Workshop on Economics and Heterogeneous Interacting Agents, Kyoto, Japan (2004)
23. Iglesias, C.A., Garijo, M., Gonzalez, J.C., Velasco, J.R.: Analysis and Design of Multiagent Systems Using MAS-CommonKADS. In: Singh, M.P., Rao, A., Wooldridge, M.J. (eds.) ATAL 1997. LNCS (LNAI), vol. 1365, Springer, Heidelberg (1998)
24. Karow, M., Gehlert, A.: On the Transition from Computation Independent to Platform Independent Models. In: Proc. of the 12th Americas Conference on Information Systems, Acapulco, Mexico (August 2006)
25. Klügl, F., Herrler, R., Fehler, M.: SeSAm: implementation of agent-based simulation using visual programming. In: Proc. of AAMAS 2006, pp. 1439–1440 (2006)
26. Lees, M., Logan, B., Theodoropoulos, G.: Distributed Simulation of Agent-Based Systems with HLA. ACM Transactions on Modeling and Computer Simulation (TOMACS) 17(3), 11–35 (2007)
27. Molesini, A., Omicini, A., Ricci, A., Denti, E.: Zooming Multi-Agent Systems. In: Müller, J.P., Zambonelli, F. (eds.) AOSE 2005. LNCS, vol. 3950, pp. 81–93. Springer, Heidelberg (2006)
28. Nebrijo Duarte, J., de Lara, J.: ODiM: A Model-Driven Approach to Agent-Based Simulation. In: Proc. of the 23rd European Conference on Modelling and Simulation, Madrid, Spain, June 9-12 (2009)
29. North, M.J., Howe, T.R., Collier, N.T., Vos, J.R.: Repast Simphony Runtime System. In: Proc. of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms, Chicago, IL (2005b)
30. North, M.J., Macal, C.M.: Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation. Oxford University Press (2007)
31. Object Management Group (OMG). Meta Object Facility (MOF) Specifications (version 2.0), `http://www.omg.org/spec/MOF/2.0/`
32. Object Management Group (OMG). Model Driven Architecture (MDA) Guide Version 1.0.1, `http://www.omg.org/cgi-bin/doc?omg/03-06-01`
33. Object Management Group (OMG). MOF Query/Views/Transformations (QVT) Specifications (version 1.0), `http://www.omg.org/spec/QVT/1.0/`
34. Padgham, L., Winikoff, M.: Prometheus: a methodology for developing intelligent agents. In: AAMAS 2002: Proc. of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 37–38. ACM Press (2002)
35. Parker, M.T.: What is Ascape and Why Should You Care? J. Artificial Societies and Social Simulation 4(1) (2001)
36. Pavón, J., Gómez-Sanz, J.J., Fuentes, R.: The INGENIAS Methodology and Tools. In: Agent-Oriented Methodologies. pp. 236–276. Idea Group Publishing (2005)
37. Pavon, J., Sansores, C., Gómez-Sanz, J.J.: Modelling and simulation of social systems with INGENIAS. Int. J. of Agent-Oriented Software Engineering 2(2), 196–221 (2008)
38. Schauerhuber, A., Wimmer, M., Kapsammer, E.: Bridging existing Web modeling languages to model-driven engineering: a metamodel for WebML. In: Proc. of the 6th Int. Conference on Web Engineering (ICWE 2006), Palo Alto, CA. ACM Press (2006)
39. Sierra, C., Sabater, J., Agusti, J., Garcia, P.: Evolutionary Programming in SADDE. In: Procedings of the First International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2002, Bologna, Italy, July 15-19, vol. 3, pp. 1270–1271. ACM Press (2002)

40. Streltchenko, O., Finin, T., Yesha, Y.: Multi-agent simulation of financial markets. In: Kimbrough, S.O., Wu, D.J. (eds.) Formal Modeling in Electronic Commerce. Springer (2003)
41. Topçu, O., Adak, M., Oğuztüzün, H.: A metamodel for federation architectures. ACM Transactions on Modeling and Computer Simulation (TOMACS) 18(3), 10–29 (2008)
42. Wagner, G.: AOR Modelling and Simulation: Towards a General Architecture for Agent-Based Discrete Event Simulation. In: Giorgini, P., Henderson-Sellers, B., Winikoff, M. (eds.) AOIS 2003. LNCS (LNAI), vol. 3030, pp. 174–188. Springer, Heidelberg (2004)
43. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems 3(3), 285–312 (2000)
44. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing Multiagent Systems: the Gaia Methodology. ACM Trans. on Software Engineering and Methodology 12(3), 317–370 (2003)