Model-driven agent-based simulation: Procedural semantics of a MAIA model

A. Ghorbani *, G.P.J. Dijkema, P. Bots, H. Alderwereld, V. Dignum

Faculty of Technology, Policy and Management, Delft University of Technology, Netherlands

A R T I C L E   I N F O

Article history:
Received 20 February 2014
Received in revised form 25 July 2014
Accepted 31 July 2014
Available online 27 August 2014

Keywords:
Model-driven simulation
Modelling language
Model-to-code
Agent-based modelling

A B S T R A C T

Agent-based modelling and simulation (ABMS) is highly instrumental for studying socio-technical systems. MAIA – Modelling Agents using Institutional Analysis – is an ABMS modelling framework that formalises social sciences knowledge. It enables handling the complexity of large complex systems, allows collaborative model development and the reuse of model components when building simulations. We detail the procedural semantics for transforming a MAIA model into an executable simulation. Its evaluation through various case studies of model development and simulation is described. The MAIA metamodel is a declarative language to conceptualise an ABM. A model description in MAIA thus provides sufficient information to translate it into a simulation model – it defines the agents, their decision-making process and their actions, all within an institutional and physical context, to affect system states that are defined in the MAIA model. A modeller can use MAIA to specify and document her model and to build her simulation by using MAIA’s semi-automatic code generation option.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Socio-technical systems, i.e., social systems that are intertwined with technology, consist of interacting heterogeneous decision making entities and technological artefacts. These systems are the subject of policy problems which explore the long term effect of strategic decisions on the operational behaviour of individuals and on the global outcomes of the system. In search for effective policies, scientists and policy analysts try to understand the system through various approaches, ranging from benchmarking and historical analysis [37] to computational modelling and simulations [20].

A simulation imitates the operation of a system over time, to show how it evolves [4]. One simulation approach that is particularly insightful for studying systems at the level of individual behaviour is agent-based modelling and simulation (ABMS). In ABMS, agents are the active entities who are scheduled to perform operations in a given space [35]. A model provides the conceptual description of the agents, actions and space that together represent a system from a specific viewpoint. The simulation is the result of executing the model in a computer. Modelling and simulation are recognised as two different stages of ABMS. However, there is no consensus about the transition between these two stages where modelling ends and simulation begins [35].

To build the model and run the simulation, making simulations of large complex systems requires common modelling languages and standardised procedures.

* Corresponding author. Tel.: + 31 15 278 6703.
E-mail address: a.ghorbani@tudelft.nl (A. Ghorbani).

http://dx.doi.org/10.1016/j.simpat.2014.07.009
1569-190X/© 2014 Elsevier B.V. All rights reserved.
For small systems and simple phenomena one can use equations or diagrams to build computational models. As illustrated in part (a) of Fig. 1, for such systems, there is no need for higher level conceptual models. Low level programming languages may be used to implement the models and build running simulations. However, modelling becomes more difficult and even impossible as the complexity of the systems and simulations grow, which requires the management of numerous concepts and relations.

For this, the modeller can make a description of the model in pseudocode (e.g., [15]), use diagrams (e.g., [27,29,9,5]), use equations (e.g., [8,41]) or deploy a generic modelling language such as UML (e.g., [2,23,6]). Subsequently she translates her description into a simulation, not using any predefined procedure. When larger models are developed, however, typically various parties are engaged in the development process, which gives rise to coordination problems especially when the rules for interpretation and translation are not standardised between the parties.

We conjecture that a high level modelling language can facilitate the development and maintenance of complex models, facilitate collaborative ABMS, and support share and reuse of models. The presence of a modelling language can result in a multi-party modelling process that is illustrated in part (b) of Fig. 1. MAIA (Modelling Agent systems based on Institutional Analysis), as introduced in [18], is an agent-based meta-model which has been developed for exactly this purpose. MAIA helps to structure knowledge about a given socio-technical system, and to organise concepts in order to build an agent-based model.

In order to use MAIA as a high level modelling language for building complex simulations, the definition of the MAIA meta-model, presented in [18], is not sufficient. The current definition of MAIA only supports the development of a static description of an agent-based model. To build a simulation from a model described in MAIA however, the procedural semantics of the transition between a model and a simulation is required.

The goal of this paper is to show how a MAIA model can be interpreted as a simulation. We identify the general requirements for building agent-based simulations and explain how a conceptual MAIA model can be transformed into a simulation.

The background of MAIA

The concepts in the MAIA meta-model are a formalisation of the Institutional Analysis and Development (IAD) framework of Elinor Ostrom [31], extended with concepts from other social science theories (Structuration [19], social mechanisms [22] and actor-centred institutionalism [37]).

The MAIA meta-model views a socio-technical system as bounded in time and space, and shaped by social structure [19]. The structure of the system is the means to organise the system and its actors who perform actions and interact with each other in what is called an action arena [32]. What happens in the action arena of the system leads to patterns of interaction and outcomes that are judged on the basis of evaluative criteria which are defined by the analyst [32].

The MAIA meta-model, mainly inspired by IAD [31] and an extension of concepts in OperA [13], is organised in five structures that serve as place holders for related concepts:

2. Constitutional Structure: the social context.
3. Physical Structure: the physical aspects of the system.
4. Operational Structure: the dynamics of the system.
5. Evaluative Structure: the concepts that are used to validate and measure the outcomes of the system.

The complete MAIA meta-model is presented in the class diagram in Fig. 6. In this section we will briefly explain it using the following case example.

2.0.1. Working example

This example describes a scenario where people make decisions about the method of care for their elderly family members. The goal is to find out which incentives result in more people taking care of their family instead of sending them to care homes or hiring nurses. Means of transport, the health situation of the parent and price of care homes are factors that affect the decision of the individuals. We model the following policy options to study their influence: (1) tax return to those people who take care of their elderly families, (2) extra days off and flexible working days or (3) higher taxes on care homes.

This working example is chosen for simplicity in order to explain MAIA concepts and the simulation development procedure. We will introduce the actual cases MAIA has been applied to, in Section 5.

---

1 A meta-model is a set of concepts and relations highlighting the common properties of a class of models. It is used to define the syntax of a modelling language at an abstract level [24].

2 More about MAIA and the modelling environment can be found at maia.tudelft.nl.
2.1. The collective structure

The collective structure describes the characteristics of the collective unit of interest (e.g., families). It defines agent types which represent individual or composite entities that make decisions, act, and react in a socio-technical system.

Agents have properties (e.g., age and gender), personal values (e.g., social recognition, wealth), physical assets (e.g., car) and information (e.g., care home fees). Agents perform various actions. Since they are social entities, most of these actions are related to the roles (e.g., elderly, child) they take in the system. Nonetheless, agents have intrinsic capabilities such as eating and sleeping that are independent of the role they take in the society. The decision making procedure of agents for performing various actions is based on these attributes (explained in Section 3.3).

2.2. The constitutional structure

Agents take roles in a socio-technical system which places them in an institutional setting. The agents must meet the condition to enact a role (e.g., age > 70 to be in the role of an elderly). Each role is created to serve an objective in the system. For example, the objective of the elderly role is to live independently. For an agent enacting a role, certain responsibilities or capabilities become available or acceptable to perform. For example, the agent in the role of an elderly can request for care, and an agent in the role of a child has the responsibility to take care of his elderly parent. These examples also imply that taking roles in a social system creates dependencies between roles (e.g., an elderly person depends on her/his child for help).

Social rules, norms, or strategies defined as institutions [32], shape and influence agent behaviour and decision making process. In MAIA these concepts are defined as ADICO institutional statements [11]. Table 1, shows some examples of ADICO institutional statements in the family care setting. When an institution has an explicit and unique sanction (i.e., ‘or else’) for non-compliance, the statement is referred to as a ‘rule’. If a sanction does not exist or is not clear and unique, the statement is called a ‘norm’. When agents follow similar strategies without being obliged to do so and there is no sanction for non-compliance to the common behaviour, that strategy is called a ‘shared strategy’ or a routine.

2.3. The physical structure

Individuals are also influenced by their physical surroundings. Physical components are the building block of the non-social environment that the agents are embedded in (e.g. care home and its room).

The elderly parent, for example, may own a house and the child may have a car or a bicycle to visit his elderly parent. The type of these components is private because they belong to a person or a group of people. Physical components can also be shared among everyone in the system (public), such as a train, or care home. Each physical component may have properties

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Deontic</th>
<th>Aim</th>
<th>Condition</th>
<th>Or else</th>
<th>Institution Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child</td>
<td>Must</td>
<td>Pay for parent</td>
<td>Parent no money</td>
<td>Parent out</td>
<td>Rule</td>
</tr>
<tr>
<td>Child</td>
<td>Must</td>
<td>Visit parent</td>
<td>Every day</td>
<td>–</td>
<td>Norm</td>
</tr>
<tr>
<td>Child</td>
<td>May</td>
<td>Claim tax return</td>
<td>Take care of parent</td>
<td>Can appeal</td>
<td>Rule</td>
</tr>
<tr>
<td>Child</td>
<td>May</td>
<td>Claim days off</td>
<td>Take care of parent</td>
<td>Can appeal</td>
<td>Rule</td>
</tr>
<tr>
<td>Employer</td>
<td>Must</td>
<td>Give days off or return tax</td>
<td>If employee claims</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Child</td>
<td>–</td>
<td>Send its elderly to nursing home</td>
<td>Parent has alzheimer</td>
<td>–</td>
<td>Shared strategy</td>
</tr>
<tr>
<td>Child</td>
<td>–</td>
<td>Visit grandparent</td>
<td>Parent busy</td>
<td>–</td>
<td>Shared strategy</td>
</tr>
<tr>
<td>Child</td>
<td>–</td>
<td>Hire a nurse</td>
<td>Parent is physically disabled</td>
<td>–</td>
<td>Shared strategy</td>
</tr>
</tbody>
</table>
(e.g., a care home has capacity and location). Physical components may also have behaviours (e.g., ageing) and affordances (i.e., what can be done with it; e.g., a bicycle can be ridden).

A composition relationship (i.e., ‘has-a’) may exist between physical components. For example, a house has rooms. To model spatial elements, MAIA also defines the connection between physical components (e.g., between houses).

2.4. The operational structure

The operational structure is an action arena where different situations take place, in which participants interact as they are affected by the environment and produce outcomes that in turn affect the environment.

The agents, influenced by the social and physical setting of the system, perform actions in the action arena. The action arena contains all the entity actions, that may execute during a simulation ordered by plans which are in turn ordered by action situations.

Entity actions have an action body which is the actual activity the performer executes. Each entity action specifies the preconditions for the performer to perform an action (e.g., IF distanceToElderly = medium & child.hasCar, visitParent 2 per day) and the updates in the status of the system after the action is executed (e.g., elderly parent at care home).

The performer of an entity action can be a physical component, an agent or a role. We assume that only agents and physical components execute actions. If the performer of the action is a role, the agent who enacts that role has the condition to execute the action. This condition is checked in the precondition part of an entity action. Actions that are associated to physical components are treated in two different ways. If an action is an affordance (e.g., car driven), the agent who owns the physical component will use that component to perform the action (e.g., drive car). If the action is a behaviour of physical component (e.g., get flat tire), the action is executed by the physical component.

The agent may have a decision making criterion for performing an action (do I send my parent to a care home?) which may also be influenced by a related institution (e.g., a norm: do not send your parent to a care home if she does not have mental problems.).

2.4.1. Planning actions

The modeller specifies the order of action execution in the Action Arena which consists of a set of action situations. The order of action execution can be specified by four different types of plans which constitute the action situations.

Given plans $p_1$, $p_2$, $p$, entity action $a$ (as an atomic plan), ';' as sequence of plans, '?' as choice and 'loop' as a conditional loop we define a plan according to the following:

- Atomic ($a$): These plans contain only one entity action (e.g., feed elderly).
- Sequence ($p_1 ; p_2$): Several actions execute one after another.
- Choice ($p_1 ? p_2 ? p_3$): By default choice plans provide an equal probability alternative between a list of actions (e.g., have breakfast or take shower).
- Loop (loop(condition $\perp p$)): A loop plan is repeated while the condition holds.

2.5. The evaluative structure

The implementation of any kind of software, including a simulation, is prone to errors which should be detected as early as possible starting from the analysis and conceptualisation phase [3]. Observed discrepancies between simulation outcomes and reality may be due to software bugs, but also to conceptualisation errors [36]. The Evaluative Structure provides concepts with the help of which the modeller can indicate what patterns of interaction, evaluation, and outcomes she is interested in. The modeller identifies those variables that can serve as indicators for model validity (is it sufficiently realistic?) and model usability (will its implementation help me to explore the question(s) I set out to address?).

Agent-based simulations should be able to track system states and performance in order to measure the outcomes of policies options (e.g., Which policy works best in creating incentives for people to take care of their elderly?). The variables that are defined to track the states of the system and its performance are called problem domain variables in MAIA (e.g., number of old people, number of old people in care home, number of old people with hired nurse) which may be related to variables in the model (e.g., death rate, care home capacity). Another set of variables called validation variables are used for debugging and validating the model (e.g., number of deaths).

3. Generating simulations from a MAIA model

By building a conceptual model, the modeller has specified a collection of concepts and relations that provide a static description of the system that is to be simulated. In this section, we explain the procedural semantics of building a simulation from a MAIA model.

The first step a programmer would need to take to build a simulation from a MAIA model is to initialize the agents, the physical components and the variables in the evaluative structure. With a MAIA model, these attributes (e.g., agent.age) are

---

3 This section is a revised version of the content presented in [17].
defined without any value so that the programmer can initiate the model with any given dataset for different experimental set ups. The number of agents, physical components and the period of the time loop are not provided by the conceptual model either because of the same reason.

Agent-based simulations have a time loop where the initialized agents perform and possibly repeat actions over time. Within the time loop, the programmer first makes a routine procedure called \texttt{UPDATE ROLES-ETC} to perform several default tasks such as assigning roles to agents. There may also be other case related requirements that can be coded in this procedure. For example, in most cases, the experimental variables of the evaluative structure also need to be updated to store new data from the previous simulation round. It is also possible to implement another \texttt{UPDATE} method inside the agent loop for certain parameters that need update per agent execution. After the update procedure, the action arena is repeated for each agent and physical component in every time step (i.e., tick) (Listing 1).

3.1. Action arena

Fig. 2, shows how agent instances enter the action arena during the simulation. Each agent instance may own a physical component (e.g., car) as he enters the action arena (illustrated by a square next to the agent).

The order of actions in the action arena is specified by the modeller with the plan concept in MAIA (see Section 2.4.1). Each atomic action in a plan is considered as an individual step. At the beginning of the simulation, the agent starts from the first action in the first plan of the first action situation. Plans (\texttt{planId}) and the steps (\texttt{stepId}) in each plan are uniquely identified for each agent instance. In later time steps, the agent resumes execution by starting from the point he had left in the previous round, indicated by these two ids.

According to the type of plan, the agents will resume differently. In the case of a loop plan, the agent will only continue the actions inside the loop until all loop conditions related to his current position in the action arena hold. Otherwise, he will return from the loop plan and update his location in the action arena to point to the plans following the loop plan. This situation is explained with an example in Fig. 3. If an agent’s planId is pointing to P24, he checks all the loop conditions that are related to his current position in the arena which is in this case only P12. If the condition for P12 does not hold, he will exit P12 and instead of continuing with P25, he will go to P13.

As soon as the agent finds an action that he may perform (i.e., preconditions meet), he either executes the action or \textit{decides} (explained later) not to perform that action. He will then leave the action arena even if he has decided not to perform the

\begin{verbatim}
1 BEGIN SIMULATION
2 INITIALIZE
3 BEGIN LOOP (time)
4     UPDATE ROLE_ETC
5     FOR EACH agent AND FOR EACH physical_component
6         ACTION ARENA
7 END LOOP
8 EVALUATE RESULTS
9 END SIMULATION
\end{verbatim}

\textbf{Listing 1.} The simulation.
3.2. Agents in action

Actions are static descriptions in the action arena that agents enter, act in and exit. An agent checks the possibility to execute an action, by first checking the preconditions for performing that action. If these conditions are met, he will act in different manners that depend on whether there is an institution or decision making procedure associated to the action.

Based on the definition of institutions in Section 2.2, there are three different types of institutional actions. Listing 2 presents a rule-based action execution procedure. When the preconditions for the action hold, the agent checks the institution conditions. When these also hold, the agent makes a decision about executing or not executing that action. For example,
when an agent is in the ‘pay care home fees’ action, he checks the precondition (parent.inCareHome), then he checks the institution condition (isEndOfMonth). The agent then decides about paying the care home or not. As we will explain in the next section, agents may decide to ignore institutions. Such disregard of rules, norms and even shared strategies is recorded. The reason to record this is to (1) enable sanctioning mechanisms and (2) use this behaviour to influence future decisions regarding non-compliance (i.e., feeling of shame) (cf. [10]).

While making a decision, the agent considers the deontic type of the institution. If an agent executes an action that he is prohibited to perform, or does not execute one that he is obliged to perform, he faces a sanction. For example, if an agent decides not to pay for the care home, the care home moves the elderly person out. Such non-compliance raises a flag so that the agent executing the sanction (the care home) can identify this agent as one that has not complied. Therefore, the care home is permitted to move out an elderly person.

The objective of the role the agent is enacting will update after the decision making process. In the MAIA meta-model, the institutions associated with a role are assumed to be in line with the objectives of that role. To enforce this, by default, for every role that the agent enacts, we define an attribute called ‘objective distance’ that reflects how far the agent is from reaching that particular objective. The agent’s compliance with institutions decreases the value of this attribute. When agents disobey norms, the objective of the agent for the associated role is updated (i.e., the ‘objective distance’ is increased) to reflect the idea that non-compliance to institutions takes agents further away from meeting the objective of the role they are enacting.

At the beginning of each tick (Listing 1: update role-et-etc), the agents check their ‘objective distances’. If the value is above a specified threshold, the agent increases his ‘willingness to comply’ which is another default attribute of the agent defined per each role the agent enacts. An agent’s ‘willingness to comply’ value partly determines how much he complies with institutions in his decision makings as we will explain later. The ‘willingness to comply’ and ‘objective distance’ attributes are a simplified learning mechanism to consider the consequence of non-compliance with institutions on the future behaviour of agents.

There are three types of institutional actions: rule-based actions, norm-based actions and actions based on shared strategy. A norm-based action and an action based on shared strategy follow similar procedures to the one explained above, except that there is no sanction for a norm-based action and no deontic nor sanction for a shared strategy.

An action can have more than one associated institutional action. If there is a list of shared strategies, the agents follow a shared strategy procedure. If there is no rule in the set of institutions for an action, but at least one norm, the norm-based procedure is executed. Otherwise, if there is at least one institutional rule in the list of institutions for an action, a rule-based procedure is executed and the conditions of all institutions are checked.

An action may have no associated institutions. In that case, the agent may still go through a decision making process before executing an action. If the preconditions to execute an action hold, the agent performs the action only if the result of his decision making process is also positive. The body for this type of action, can be an institutional responsibility of a role (e.g., takeCareOfParent() for the child role), or an intrinsic capability of an agent (e.g., eat()). Note that in the precondition of the action with institutional capability, the agent checks whether he has the associated role (e.g., IF agent.isParent()). Finally, an action may also have no decision making process and no associated institutions. For this simple form of action, the agent executes the action by only checking the preconditions (e.g., IF agent.isAlive THEN breath()). This type of execution is used if the body of that action is an intrinsic capability of an agent (e.g., eat()) (see Section 2.1) or a behaviour of physical component (e.g., getFlatTires()) (see Section 2.3).

3.3. Agents decision making

The default decision making algorithm in MAIA is Multi-Criteria Decision Making (MCDM) [21] presented as pseudo code in Listing 3. In each decision making routine, the agents check a number of conditions (optional to define in the conceptual model). In the family example, the agent checks whether the parent has been in a care home before and if the parent’s age is above a certain value. The agent also has a number of criteria that affect his decision. These decision aspects have numerical values which are prioritized with weights. These aspects can be properties of the agent, other agents, physical components or global variables among others. By providing numerical values for sanctions (e.g., impose a fine), the agent can also take the sanctions into account. The outcome of every decision making process is a boolean value: if the weighted sum of aspects is larger than a specified threshold and the conditions hold, the agent decides to perform the action.

```
1 BEGIN MCDM(agent)
  IF conditions
  weightedSum = Aspect1 * Weight1 + ... + AspectN* weightN
  IF weightedSum > decision-threshold
  RETURN TRUE
  END IF
  END IF
RETURN FALSE
END MCDM
```

Listing 3. Multi-criteria decision making algorithm (cf. Fig. 6).
Since the outcome of a decision is binary, agents cannot explicitly choose from a list of actions. Therefore, if an agent must choose an action from a list of possible actions, he has to weigh each option separately. In the family care example, we want the agent to choose between (1) send to care home, (2) hire nurse, and (3) self care for the elderly. Each of these are separate actions with separate decisions. However, when the agent is deciding whether to put his elderly parent in a care home, he also considers the cost of hiring nurse or tax return benefits while making the decision on care home. In other words, even though we have a boolean outcome for each decision, we may still implicitly consider the consequence of performing other actions in one decision.

Unlike MCDM aspects that need to have numerical values, decision conditions provide more flexibility for defining decision influencing factors. The condition part of a decision and the aspects can be defined interchangeably in the conceptual model (e.g., the age of parent can be considered as a condition \((\text{age} > 60)\) or an aspect \((\text{age} \times \text{weight})\)).

The ‘willingness to comply’ attribute, introduced previously, affects the decision when there is an institution associated to the action. For example, if a norm tells the agent to visit his parent every day, his ‘willingness to comply’ value among other factors, will influence the outcome of the decision about visiting his elderly parent.

In the decision making procedure to perform an action with several institutional statements, priority (higher weight) is given to an institution with sanction (i.e., rule) \((\text{rule} \gg \text{norm} \gg \text{shared strategy})\). So there is, by default, a higher probability of obeying a rule that has a sanction.

3.4. The simulation environment

The simulation environment represents the physical aspects of the simulated system and has two aspects. It is a placeholder for public physical components and it specifies the visualisation of the simulation (if required).

The public physical components (see Section 2.3) are shared between all agents in the model. The status of these components is updated according to the actions agents perform on them. If a simulation requires a visual representation, each node (i.e., physical component) must have coordinates as its properties. The physical connections between physical components specifies the edges of the network.

To make visual representation for agents, a physical component (i.e., body) needs to be defined for the agent during conceptualisation.

3.5. Evaluation of simulation results

The results of the simulation are captured in the main model using the variables defined in the evaluative structure of a MAIA conceptual model. In the MAIA model of the family example, we defined problem domain variables as: \(\text{numPeopleInCareHome}, \text{numPeopleWithNurse}, \text{numPeopleFamilyCare}\). We monitored these variables for different values of the independent variables: tax return, free days and tax on care homes. Fig. 4 shows some results from the family care example. In these diagrams, we can see that increasing the salary of the people or in other words, tax return, would be the most effective incentive for them to take care of their elderly parent. However, increasing salaries would also increase hiring nurses because of the extra income people receive. Therefore, the option of giving free days to people who take care of their families seems to be more effective if nurses are scarce. The analysis of such results is also aided by the relations between variables and entity actions specified during conceptualisation in the evaluative structure.

In this section we explained how a MAIA model maps into an agent-based simulation. Table 2 summarises the options available for implementing agent decision making and actions, plan options and the two different physical component implementation options. In the next section we will briefly describe the model-driven development process for this transformation practice.

4. A model-driven approach to build simulations

The transformation from a high-level modelling language such as MAIA to a simulation preferably should be in accordance with Model-driven Software Development (MDSD) practices.

MDSD is the process of building software, through a step by step procedure and by using different levels of abstraction in software specification. Since MDSD facilitates the development of a software system from a conceptual model it is a suitable approach to build software simulations from MAIA models.

The main requirements for a Model-driven Software Development (MDSD) approach are (1) meta-models that describe domain specific models, (2) transformation modules that map one type of model to another (e.g., MAIA model to Java Code), and (3) platforms that produce executable software [40].

Fig. 5 depicts a standard MDSD approach. At the conceptual level, a computational independent model (CIM) describes the domain concepts of the model, focusing on the requirements of the software system, which is the simulation in this case [30]. Since the MAIA meta-model is designed to address ‘what’ concepts to put in a simulation independent of a simulation platform, it can be considered as a CIM meta-model. Therefore, an agent-based model conceptualised using MAIA such as the family care example, is a CIM instance (i.e., domain model) of the MAIA meta-model.
A platform specific model (PSM) is one that bridges a CIM and a software platform. A CIM requires mapping modules that explain what each concept is translated to, in a PSM. 

By making the assumption that the specifications of the Java programming language (e.g., class, object, method, attribute) form a PSM meta-model, the minimum requirement to get to executable simulation code from a MAIA model is to define a mapping module that converts MAIA concepts to Java.

Section 3 presented the semantics behinds the mapping modules that can transform a MAIA model into a simulation independent of the platform. These semantics are in fact the result of implementing several case studies with MAIA following the prototyping method for identifying these modules [34]. Similar to this method, we used our case studies to recursively extract these rules. The exact Java transformations can be found in [17].

The more specific transformation between MAIA and Java is conducted using a set of templates called Java Emitter Templates (JET), which are a technology to produce Java code from an E-core model in Eclipse EMF [14]. We implement 9 main JET templates:

<table>
<thead>
<tr>
<th>Decision making</th>
<th>Actions</th>
<th>Plans</th>
<th>Physical components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition checking</td>
<td>Non-institutional</td>
<td>Loop</td>
<td>Public</td>
</tr>
<tr>
<td>MCDM</td>
<td>With decision making</td>
<td>Sequence</td>
<td></td>
</tr>
<tr>
<td>MCDM and condition checking</td>
<td>Without decision making</td>
<td>Choice</td>
<td>Private</td>
</tr>
<tr>
<td>MCDM and condition checking</td>
<td>Institutional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCDM</td>
<td>rule-based</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCDM</td>
<td>Norm-based</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCDM</td>
<td>Shared strategy-based</td>
<td>Atomic</td>
<td></td>
</tr>
</tbody>
</table>

A platform specific model (PSM) is one that bridges a CIM and a software platform. A CIM requires mapping modules that explain what each concept is translated to, in a PSM.

By making the assumption that the specifications of the Java programming language (e.g., class, object, method, attribute) form a PSM meta-model, the minimum requirement to get to executable simulation code from a MAIA model is to define a mapping module that converts MAIA concepts to Java.

Section 3 presented the semantics behinds the mapping modules that can transform a MAIA model into a simulation independent of the platform. These semantics are in fact the result of implementing several case studies with MAIA following the prototyping method for identifying these modules [34]. Similar to this method, we used our case studies to recursively extract these rules. The exact Java transformations can be found in [17].

The more specific transformation between MAIA and Java is conducted using a set of templates called Java Emitter Templates (JET), which are a technology to produce Java code from an E-core model in Eclipse EMF [14]. We implement 9 main JET templates:

---

4 A platform can be considered as a programming language (e.g., Java), programming platform (e.g., Eclipse for Java), or a simulation platform such as Repast.

5 Because it is possible to define many CIMs and PSMs at different levels in the MDSD process, we can even consider the Java Language as a PSM meta-model. However, one may also argue that the conversion between MAIA models and Java Simulations is direct transformation to code (i.e., CIM to code). This is still a recognised model-driven approach as described by The Object Management Group [30].
1. **Agent**: A MAIA agent transforms into a Java `Agent` class. All agent classes in the model extend the abstract `Agent` class. For each agent, we define a boolean for every possible role that the agent can take (is ‘Role’, e.g., `isChild`). All the physical components (type: private) that the agent owns are also instantiated for the agent. The properties and personal values are all defined as attributes for agent classes. The abstract `Agent` class has `plandId`, `stepId` and `willingnessToComply` as default attributes.

2. **Role**: Roles extend the abstract `Role` class. Roles are static classes in Java. They have a method `entryCondition()` where the agents check whether they can or cannot enact a role.

3. **Objective**: For each role, an `Objective` class is implemented (e.g., `Child-Objective`) that will be instantiated for each agent that enacts the role. This class has an `objectiveDistance` attribute and a method that calculates whether the agent has reached his objective.

4. **Institution**: Similar to roles, institutions are also static. Each institution class has a `deonticType` attribute and a `condition()` method.

5. **Decision Making**: A `DecisionCriterion` class has a `condition()` method, an `MCDAcalculation()` method where the actual aspects of a condition and the associated weights are put into a formula, and a `result()` method. In the `result()` method, the output of the calculation method is checked against a specified `threshold` returning a boolean value.

6. **Physical Component**: Depending on the type of physical component, the classes can be instantiated (type: private) or defined as static classes. where they can be used from any other class in the simulation. If there are composition relations between physical components, the component class that contains the other components will have an instance of the component (e.g., A car has four instances of wheels). A physical component JET-template is illustrated in Listing 4.

7. **Action**: Entity actions are represented as action classes in Java, all extending the abstract `Action` class. Each entity action class has a `preCondition()` method and an `execute()` method which is similar to the pseudo-code presented in Listing 2.

---

**Fig. 5.** Model-driven software development.

**Listing 4.** A JET template for physical component instances.
8. **Action Arena**: The main class of the Java project calls the `ActionArena` class. Based on the plan specifications, in the run method of the `ActionArena` class, the agent who enters the method, executes the actions one at a time.

9. **MainSimulation class**: The main simulation class has three methods: an initialization method, a scheduling method which contains the time loop and agent physical component lists, and an evaluation method.

The JET templates create a Java project from a MAIA model. In order to run the simulation, the Java code requires further completion, for example, by adding initial values, the period of the simulation and the number of agents.

Besides the JET templates, there are other options for transforming a MAIA model into executable code, for example, by adding a **platform independent model** (PIM) to the MAIA development process [30]. Defining a PIM meta-model can in fact make the transformation of a MAIA model into a simulation platform-independent, so that MAIA models could be translated to any of the already existing ABMS platforms such as Repast, Netlogo, MASON etcetera.

AMP (Agent Modelling Platform) provides a platform independent meta-model called ‘Agent Modelling Framework’ (AMF) [1]. A model conceptualised in AMF can so far be translated into simulations in Repast, Escape and Ascape platforms. AMF is pluggable and modular so that other developers can develop AMF generators for their own tools (e.g., Netlogo) [1]. The MDSD procedure of simulation generation from MAIA using AMP is explained in [17].

Using the MDSD method, MAIA models are converted to executable simulations with the information provided in this paper. Although this process can be done (semi) automatically, the information can also be used as guidelines for programmers in order to build simulations.

### 5. Evaluation

The goal of this paper was to show how an agent-based simulation can be developed using a model conceptualised with the MAIA meta-model. The meta-model itself and the procedure described in this paper have been evaluated, using various evaluation methods in the literature, which we will briefly discuss in this section.

1. **Case study** [25]: MAIA has been used to build agent-based models for real world applications.
2. **Survey** [25]: Using questionnaires, the feedbacks from the users of MAIA were used to evaluate this tool and to further improve it.
3. **Formal experiments** [25]: Two of the ABMS projects were developed with and without MAIA to evaluate its benefits and limitations.
4. **Comparison with similar tools** [28]: A set of features and functionalities were defined as the basis for comparison with other tools.

#### 5.1. Case studies

The MAIA project is a methodological project which aims to bring ABMS within the reach of social scientists or practitioners who have less familiarity and experience with modelling. Therefore, the first question that rises from our methodological work is whether MAIA is actually bringing ABMS within the reach of modellers. To find out, we used 7 case studies and asked several social scientists who were mostly inexperienced modellers to build an agent-based model for their case. Consequently, the users were asked to fill in questionnaires, providing feedback on the usefulness and usability of MAIA. The modelling projects not only reflected the usefulness of MAIA, they also helped us define the procedural semantics of making simulations as presented in this article. We will briefly introduce each of these case studies.

**5.1.1. Consumer lighting transitions**

The goal of this project was to identify policies that would be more effective in the transition of consumer lighting to more efficient lighting (i.e. LED) products. Three possible policy interventions were explored: (1) a ban on incandescent bulbs, (2) an taxation scheme on incandescent bulbs, and (3) a subsidy scheme on LED lamps [17].

It is worth noting, that as part of the formal experiment, this model we conceptualised with and without MAIA.

**5.1.2. The wood-fuel market in Switzerland**

This project aimed to understand the influence of different factors (e.g. demand, fossil energy prices, co-product markets, policy measures or natural disasters) on the availability of wood-fuel to foster wood energy [26]. Similar to the previous case study, this model we also developed with and without MAIA.

**5.1.3. Backyard E-waste recycling in Bangalore, India**

This project was set out to identify those factors that influence the transition of the informal recycling sector in Bangalore, India into a system where informal waste collectors and recyclers cooperate with professional end refiners, to create a recycling and recovery economy that leads to less environmental and health hazards [39].

---

*Details of the user evaluation can be found in [17].*
5.1.4. Bio-gas energy system in the Netherlands
This project consisted of 3 separate agent-based models. The first model was designed to identify those factors that influence the development of a manure-based energy system within rural regions in the Netherlands from a farmer perspective. The second and third project studied whether horizontal governance is effective among biogas producers (waste water, manure) and consumers [12]. The final two agent-based models were in fact developed using the same MAIA model. However, since the focus of each of them were different, the type of analysis, the input data and some of the implementation details differed [42].

5.1.5. Innovation practices in the Westland horticulture sector
The goal of this agent-based model was to study the effects of social institutions on innovation practices in the Westland horticulture sector [38].

5.2. Comparison with related work
Since MAIA is in the tool category of the ABMS literature, we compared its functionalities with other modelling tools and frameworks. To perform this comparison, we view an ABMS tool in terms of software functionalities (e.g., modelling language), agent related aspects (e.g., autonomy of agents, sociability) and simulation functionalities (e.g., time loop). The complete list of features can be found in [17]. Besides the overall evaluation, we compared MAIA with INGENIAS and easyABMS which are the most comparable ABMS tools in the literature in terms of the functionalities and features they provide.

easyABMS [16] is an agent-based simulation framework that is connected to the Repast platform. It follows a model-driven approach and uses UML diagrams (in addition to its own graphical notations) in the analysis, conceptual and design phases of model development. INGENIAS [33] is a agent-based software engineering platform which has been extended to also support simulation development. Similar to easyABMS, INGENIAS has self-defined graphical notations and diagrams, and supports code generation [33]. INGENIAS has several simulation functionalities (e.g., Scheduling). This tools is also connected to Repast [35].

We will briefly compare MAIA with easyABMS and INGENIAS to show the suitability of each platform for different simulation requirements.

5.2.1. Software engineering features
While all three tools address software development requirements, they are relatively different in strength. INGENIAS covers the most complete software development cycle, starting from conceptualisation all the way to automatic code generation. Nonetheless, since it is a relatively complex tool, there is a steep learning curve. However, if the modeller knows the Repast platform, easyABMS may be a more efficient solution. MAIA enables team work in the development phase, which is especially suitable for large real world projects, where stakeholder involvement is essential. If a programmer is present in the simulation development team, as a result of a richer and extensive meta-model, using MAIA would allow more conceptually complex and advanced simulations.

5.2.2. MAS features
MAIA covers more agent related concepts (e.g., sociability, roles, organisation, autonomy) than the other two tools making it suitable for problems where the social aspects of a system are being analysed or the goal is to compare policies. However, INGENIAS has agent communication features which may be suitable for distributed agent simulations. Likewise, since easyABMS is built on top of the Repast platform, it facilitates concurrency management which is also a requirement for distributed simulations.

5.2.3. Simulation features
MAIA has several benefits compared to the other two tools. It allows the modelling of the environment, gives the possibility to build multiple societies, supports ontology development and finally, it is possible to statically (i.e., at conceptual level) validate the model by communicating with stakeholders. INGENIAS has better tool support, allows intervention in behaviour during run-time and supports data handling. easyABMS however is a better tool for data handling as it relies on Repast, it models the environment and provides dynamic validation, again relying on Repast.

In this section, we briefly explained parts of the evaluation process MAIA has gone through. Of course, MAIA is an ongoing project aiming to cover more features as an ABMS platform.

6. Discussion
The concepts in the MAIA meta-model can be used as a declarative language to conceptualise an agent-based model. However, these concepts entail dynamics. In this paper, we have described the procedural semantics that lead to the dynamics. To arrive at this description and achieve a programmable solution for making this transition, we faced several challenges that are discussed in this section.
6.1. Flexibility to disobey institutions

Since MAIA conceptualises actors as agents who are able to disobey institutions, weighing them against their own personal values, the simulation must be able to display this behaviour. The way institutions are built into actions, allows agents to consider institutional conditions and the type of institution while making decisions to perform actions.

More specifically, we assume that agents have personal values which may be sometimes conflicting with the institutions and the objectives of the roles they enact. To make a link between these two potentially conflicting attributes, we defined \textit{WillingnessToComply} as a characteristic for each agent reflecting his obedience and \textit{RoleObjectiveDeviation} to consider the fact that non-compliance affects future decisions regarding institutions. This behavioural mechanism takes various driving forces for human decision making into account and enables the simulation of apparent socially irrational behaviour. This simple implementation of learning behaviour is based on the institutional and behavioural studies of [31,19] and can therefore, represent many of the rule breaking behaviours in social systems.

6.2. Performing tasks

To categorise and organise the parameters, the agents have to consider for making decisions, we made the assumption that the agents make separate binary decisions about every action. This means that the agents do not have the capability to choose from a set of actions. However, by allowing modellers to build complex decision making procedures if required, they are still able to take the outcomes of other actions into account while making a binary decision for one action.

6.3. Coordination

Coordination and coordination problems in ABMS such as deadlocks or conflicts are issues that are extensively addressed in the literature (e.g., [7]). The procedural semantics presented in this paper provides the basics (i.e., step id, plan id and the flag concept) for enabling coordination between agents. The collective tasks can be defined as one atomic action such as lifting a table, or a plan consisting of several actions such as a negotiation procedure.

7. Conclusion and future work

The ability to build simulations from high level concepts such as the ones captured with MAIA facilitates the development and maintenance of complex simulations and supports modellers with less experience in programming to build simulations. By building a conceptual model and using the procedure explained in this paper, the modeller can build simulations using semi(automatic) code generation software or by including programmers in the simulation process.
In this paper, we explained how an agent-based model, described in the high level declarative language, MAIA, can be interpreted as a simulation. With the procedural semantics, we showed that a model description in MAIA provides sufficient information to translate into a simulation model. To summarise, a MAIA model describes the agents, their decision making process and their actions all within an institutional and physical context which results in changes in the status of the system that is also defined in the MAIA model. The only additional requirement to run a simulation is, thus, the initial values of parameters.

The procedural semantics presented in this paper is one way of translating a MAIA model into an executable simulation that involves many implementation choices and assumptions. By varying the translation protocols, there are many other possibilities for building executable simulations using MAIA as a high level modelling language. For future research, we will be further developing the software modules to facilitate fully automatic translation of MAIA models into executable simulations.

Appendix A

See Fig. 6.

References