ON THE FORMAL ANALYSIS OF THE DYNAMICS
OF SIMULATED AGENT SOCIETIES

A.E. Eiben\textsuperscript{1} \quad C.M. Jonker\textsuperscript{1,2} \quad V. Popova\textsuperscript{1} \quad M.C. Schut\textsuperscript{1}

\textsuperscript{1}Artificial Intelligence Section
Department of Computer Science
Vrije Universiteit Amsterdam
The Netherlands
{gusz, popova, schut}@cs.vu.nl

\textsuperscript{2}Nijmegen Institute for Cognition and Information,
Radboud Universiteit Nijmegen
The Netherlands
C.Jonker@nici.ru.nl

Received 25 April 2005
Revised 20 January 2006

To analyze emergent behavior, a formal framework is needed to characterize the structure and dynamics of complex interaction-based multi-agent systems. We introduce an extension of an existing agent testbed for artificial societies making it possible to formally analyze the dynamics of the simulated agent system. The extension generates temporally annotated logical terms that describe parts of the dynamics of the simulated system. Based on these terms, it is possible to validate hypotheses about the system on different levels of aggregation, i.e., agent, group and system level.

We present first results from a set of simple experiments in a class of artificial societies.

Keywords: formal analysis, agent societies, simulation, emergent behavior

1. Introduction

In simulation-based research, hypotheses are normally formulated in natural language. These hypotheses are then evaluated on the basis of output results obtained from simulation runs. We observe a discrepancy between expressing hypotheses rather informally and their evaluation process. While simulations are often complemented by formal descriptions of the system structure, these descriptions are used only for setting up the simulation and, hence, they are not exploited in the evaluation process.

In this paper, we propose a simulation system with which formalized hypotheses can be validated automatically on obtained simulation output results. Experimenter may benefit from formalizing their hypotheses originally formulated in natural language, so that the system can validate them in an automated fashion. This paper is a first important step towards making this possible.

The basis of our simulation system is a simulation software package and simulation language, which is extended with a component that generates a trace consisting of formal temporally annotated state descriptions. These state properties are the basis of more complex properties that are used either for (behavior) specifications or hypothesis validation. This paper looks at the latter.

As formalization and formal validation and verification techniques often bring with them a high computational burden, this puts restrictions on the scalability of our
approach. Still, in our case these restrictions are not as rigid as with formalization in general. Because we confine ourselves to checking formal properties within a given search space (i.e., the set of simulation runs), the validation process may still be costly but not necessarily impossible.

Finally, we mention that our approach is not supposed to replace either computing statistics or visualization of the output results obtained from simulation. Instead, our approach can be considered as a useful tool in addition to these established ways to analyze output data.

The work described in this paper is part of a larger work which has the goal to construct a formal framework to analyze the dynamics of complex interaction-based systems that typically exhibit emergent properties. As multi-agent systems scale up and grow towards systems with such degree of complexity, it is considered important to have a formal simulation analysis framework for getting insight into the behavior of such systems.

This paper addresses an important formalization issue in the analysis of agent societies, i.e., the formal specification and validation of the (dynamic) behaviors of agents populating such a society. Simulation is a good method to obtain insight into the particular dynamics of such behaviors with respect to dynamic and unpredictable environments. Hence, a model that marries formal and empirical investigation is thus a fundament on which more complex societal issues (social organization, adaptation, reorganization [2, 4, 16, 24]) can be considered.

Our approach relies on a number of assumptions which we formulate explicitly in the following. First we assume the existence of two formal languages: L1 which can be used to express (a part of) the behavior of the multi-agent system (MAS) into a formal trace and L2 which can be used to express an informal property into a formal property. L1 and L2 should be consistent which here means that properties formulated in L2 can be checked over a trace formulated in L1. In general L2 will be more complex than L1. We describe the languages we chose for our case study in Section 2.4.

The second assumption is that the constructed trace captures the behavior of the MAS relevant for checking the formal property. This can be in two dimensions: in time – the trace includes the relevant time frame where the behavior needs to be verified, and in expressivity – the trace captures all relevant facts and events connected to the behavior under question with respect to the formal property to be checked.

The last two assumptions address the translation from an informal to formal property. We assume that the informal property can be expressed closely enough in a formal property (i.e. an “acceptable” translation is possible) and that the chosen translation

---

indeed results in a formal property which expresses closely enough the informal property (i.e. the chosen translation is “acceptable”).

These last two points underpin a major hypothesis of the used methodologies and techniques in our work (and actually all formal validation techniques): that informal properties can be given a formal counterpart. The gap that exists between natural language properties and their formalization is definitely not one that can be ignored. We recognize the very fundamental question whether informal properties can be (in automated fashion or manually) properly formalized. One can, by definition, not address this question theoretically (or: in a verifiable automated fashion): every informal part over which one would want to state anything within a formal system would render the whole system informal. If one approaches the problem manually, then the used method is always error-prone. In both situations, still, there is no way to verify whether a translation is 'bad' or 'good'. We mention the hypothesis here such that the reader is aware of this; still, we note that this discussion is outside the scope of the paper. However, later in the paper (Section 3) we do give some pragmatic guidelines and heuristics that are used in the methodology to come to formal counterparts of informal natural language properties.

This paper is organized as follows. In Section 2 we describe background information on agent societies, simulation and the formalization of dynamics. Section 3 gives an overview of our simulation system. In Section 4 we present the case study in artificial societies. Section 5 contains the experimental setup. Section 6 analyses the obtained empirical results. Section 7 concludes and presents future work.

2. Background

The research described in this paper is concerned with 1) complex interaction-based systems and 2) the formal description of the dynamics of such systems. By means of simulation we gain insight in the structure and dynamics of systems. This Section briefly describes existing work on agent societies, simulation and formalization of system dynamics.

2.1. Agent Societies

In line with [17], we consider agent societies to be long-lived social constructs that are inherently open systems. Agents – that have different goals, various levels of rationality and heterogeneous capabilities – may come and go at will while the society persists. The society is then a shared domain through which agents may act (cooperate) and communicate. This means that somehow the behavior of the agents is constrained, but still with relatively much flexibility and freedom (in comparison with other organizational forms). These constraints are commonly known as social laws (or norms). Much research is concerned with how the concept of social laws can be used to facilitate the construction of large-scale open agent systems in general. As such, a society may provide a formal structure upon which more complex inter-agent behaviors can be built. The formation of a society consists of defining roles, protocols and social laws, followed by determining how agents may join and leave the society. Because a society does not
prescribe the internal behaviors of agents, any societal mechanism may only describe the externally observable characteristics of the agents.

Our review of the literature on formalization and validation/verification of agent societies reveals that relatively many papers have proposed formal models of organizations [14, 18, 19, 25], but few papers explore the verification of agent societies exhaustively, with notable exceptions [1, 12]. Still, the exceptions have one thing in common: they focus on performance. The verification process usually consists of answering questions like how much situational change it takes before the system breaks down, if the chosen mechanism can guarantee the functioning of the system, if assigned (cooperative) tasks can be appropriately executed, etc. The verification process either happens by empirical evaluation on the basis of simulation (e.g., [12] that checks whether a utility-based reorganization mechanism helps an artificial society to survive) or by logical reasoning (e.g., [1] where the possibility is checked for agents to fulfill their obligations)

This paper focuses on the validation of agent behavior underlying such performance of an organization. That means, we consider performance essential as an ultimate measurement of how well the organization behaves, but we stress that the underlying organizational structure that achieves such performance must be researched to get the bigger picture.

2.2. Simulation

The particular simulation of interest in this paper concerns one of a ‘complex interaction-based system’ – i.e., a system where the interaction between its parts determines its structure and dynamics. For defining the concepts concerning simulation, we adopt the same terminology as [23]. A system is a collection of entities that act and interact together toward the accomplishment of some logical end. A model is built as a representation of the system and is studied as a surrogate for the actual system. In situations where mathematical analysis is precluded, models may be studied by means of simulation, i.e., numerically exercising the model for input in question to see how they affect the output measures of performance.

The process of setting up a simulation study involves steps of problem formulation, data collection, model definition, experimental design, running the simulation, output data analysis and reporting of results. Throughout this process, intermediate validation steps assure that the simulation model corresponds with the actual system under investigation. The work described in this paper relates to two steps in particular, i.e., model definition and output data analysis, and we describe these in more detail.

Model definition concerns setting up a conceptual model of the actual system with respect to project objectives, performance measures, data availability, computer constraints, etc. Many tools exist nowadays to support modelers with this activity. Depending on specific user interest, one may choose from a variety of simulation languages and software packages. We provide a brief overview of such languages and packages in Section 2.3. These tools provide natural frameworks for model construction.
As such, they are based on formal system descriptions and include concepts like entities, states, events, time, variables, etc. The model definition includes validation of the simulation model: “the process of determining whether a simulation model is an accurate representation of the system, for the particular objectives of the study” [23]. Validation is essential for assuring that the simulation model corresponds with the actual system. Various validation techniques exist, of which we mention one in particular. By letting the simulation program generate a trace, i.e., the state of the simulated system (e.g., state variables, statistical counters), it is possible to compare the states with hand calculations to check the validity of the program.

Analysis of output data is in practice still rather undervalued as the simulation process is concerned. Much time goes into model development and programming, rather than addressing the generated output results appropriately. A commonly made “error” is that a single run is made of some arbitrary length, supposedly to provide insight into the workings of the actual system. Instead, suitable statistical techniques must be used to design the simulation experiments and analyze the results. Since the output processes of simulations are almost all nonstationary and autocorrelated [23], classical techniques may not always be applicable. Still, statistical techniques exist for analyzing a single system (termination conditions, mean and average estimations, etc.) and for analyzing multiple systems (measuring response differences, ranking, selection, etc.).

To our best knowledge, formal analysis and validation of the system behavior has not received much attention in the agent-based simulation modeling literature. Also, relating the output data to the structure of the model itself has not been explicitly addressed. This paper is a first step toward filling these two gaps.

2.3. Simulation Frameworks

Many agent simulation frameworks are currently around – various (survey) articles refer to dozens of such frameworks [15, 31, 33]. Various places on the internet contain overview lists of these frameworks, including download and documentation links, e.g., [32]. Some of these frameworks are more generic agent toolkits (usually aimed towards computer scientists for engineering agent-based applications), some are specifically aimed at social simulation (usually aimed towards social scientists). As for the scope of this article, we are more interested in the latter – social simulation frameworks. However, we must remark that the presented generic methodology does not particularly focus on social simulation, but rather simulation in general and therefore may also include more general agent toolkits in the future.

The mentioned surveys present overviews of frameworks based on some evaluation criterion or in historical comparison with the development of other software packages. Serenko and Detlor [31] present an overview of 20 agent toolkits and evaluate the use of these toolkits in post-secondary sources. Gilbert and Bankes [15] give an overview of approximately 10 agent-based modeling tools. It is interesting that they compare the development of these tools as observed over the last 10 years with the development of statistical software. Firstly there were implementations on mainframe computers; then
general-purpose languages were developed; following were routine libraries to support purpose-built programs; finally resulting in the development of packages containing routine collections with a common standardized user interface. For social simulation, an extensive survey was performed by Tobias and Hofmann [33]. As a result of strict overall requirements, the pool in this survey was reduced from 21 general to only 4 fitting frameworks. The requirements included, among others, that the frameworks must be java-based and suited for social simulation. Still, the 4 frameworks are thoroughly evaluated. Together, the three surveys give a good overview of agent-based modeling frameworks. For our purpose, we briefly discuss 3 frameworks in more details in this Section: Swarm, RePast and NetLogo.

One of the first agent-based modeling libraries is the Swarm framework, originally developed at the Santa-Fe Institute [26]. It is a set of code libraries to be used in agent-based simulation programs written in Objective-C or Java. The simulation comprises collections concurrently interacting agents. The use of Swarm is widespread, it is recognized by the scientific community and has a wide and strong user base; it has to be started from command line, and while a user interface can be created it is necessary to have quite some programming skills to use Swarm [33]. It supports a hierarchical modelling approach, where agents may consists of swarms of other agents in nested structures.

Another widely used framework is RePast [29]. The RePast package was evaluated best in the survey by Tobias and Hofmann [33]. It performed well on the evaluation criteria, which were general (including license, documentation, support), support for modeling and experimentation (including support for modeling, simulation control, project organization) and modeling options (including large number of complex agents, inter-agent communication, nesting of agents). RePast borrowed concepts from Swarm, but distinguishes itself along the way in that it has a number of pure implementations in several languages (java, .net and python) and built-in adaptive features (e.g., genetic algorithms and regression). It allows programmers to build simulation environments, build user interfaces and automatically collect simulation data.

We conclude with a brief description of NetLogo [3], based on the StarLogo framework [30]. It is widely used in research and education across a wide range of disciplines (epidemiology, social sciences, material sciences, earth sciences). The feature that distinguishes it from other frameworks is the HubNet, which is a technology that allows NetLogo to run distributed participatory simulations. This can be used, for example, in a classroom setting where students can run their own partial simulation and connect to other simulations. It also enables researchers to collaboratively explore a simulation. It has a community of thousands of users worldwide.

This Section overviewed a number of simulation frameworks and presented some results from recent framework surveys. For the context of this paper, it is of particular interest to look at frameworks that have a social scientific focus rather than an engineering one (i.e., directed towards engineering agent-based applications). In general, there are dozens of simulation frameworks (or agent-based modeling toolkits).
Restricting this to social simulation frameworks leaves us with approximately a dozen frameworks. Of this subset, the frameworks that are particularly suited for our purpose are those that easily communicate with other programs (for which it requires routine libraries, like Swarm) and that have a good user interface that allows well-structured and fast experimentation (for which it requires a standardized user interface, like RePast or NetLogo). A combination of two of the three discussed frameworks (Swarm and RePast, or Swarm and NetLogo) is ideal for this purpose. While the possibility of implementing such combinations remains an open question (as also concluded by Gilbert and Banks [15] and surely not trivial*), for this paper we used the JAWAS framework (discussed in Section 4.1) that we developed ourselves in the last 2 years. This allowed us to test the ideas we lay out in this paper. Still, we leave open for future work to integrate more established and wider used frameworks as discussed above.

2.4. Formalization of Dynamics

As mentioned earlier, simulation can give us insight in the structure and dynamics of the system. In this Section we discuss the formalization of dynamics that we adopt in this paper.

Dynamics will be described as evolution of states over time. The notion of a state as used here is characterized on the basis of an ontology defining a set of physical and/or mental (state) properties that may hold at a certain point in time. For example, the internal state property ‘the agent A has pain’, or the external world state property ‘the environmental temperature is 7° C’, may be expressed in terms of different ontologies. To formalize state property descriptions, an ontology is specified as a finite set of sorts, constants within these sorts, and relations and functions over these sorts. The example properties mentioned above then can be defined by nullary predicates (or proposition symbols) such as pain, or by using n-ary predicates (with \( n \geq 1 \)) like \( \text{has_temperature} \). 

For such formalization of properties, the Temporal Trace Language is used as a tool; cf. [6, Chapter 1] and [21]. For the properties occurring in the paper informal, semi-formal or formal representations are given. The formal representations are based on the Temporal Trace Language (TTL), which is briefly described in the following paragraphs.

A state ontology is a specification (in order-sorted logic) of a vocabulary. A state for ontology \( \text{Ont} \) is an assignment of truth-values \{true, false\} to the set \( \text{At(Ont)} \) of ground atoms expressed in terms of \( \text{Ont} \). The set of all possible states for state ontology \( \text{Ont} \) is denoted by \( \text{STATES(Ont)} \). The set of state properties \( \text{STATPROP(Ont)} \) for state ontology \( \text{Ont} \) is the set of all propositions over ground atoms from \( \text{At(Ont)} \). A fixed time frame \( T \) is assumed which is linearly ordered. A trace or trajectory \( \gamma \) over a state

---

* NetLogo for example allows for such communication to other programs via sockets. We are currently implementing this for another project and this is far from trivial.
ontology Ont and time frame T is a mapping γ: T \rightarrow \text{STATES(Ont)}, i.e., a sequence of states γ(t ∈ T) in \text{STATES(Ont)}. The set of all traces over state ontology Ont is denoted by TRACES(Ont). Depending on the application, the time frame T may be dense (e.g., the real numbers), or discrete (e.g., the set of integers or natural numbers or a finite initial segment of the natural numbers), or any other form, as long as it has a linear ordering. The set of dynamic properties DYNPROP(Ont) is the set of temporal statements that can be formulated with respect to traces based on the state ontology Ont in the following manner.

Given a trace γ over state ontology Ont, the input state of some role r within a group g at time point t is denoted by:

\[ \text{state}(γ, t, \text{input}(r|g)) \]

analogously:

\[ \text{state}(γ, t, \text{output}(r|g)) \]
\[ \text{state}(γ, t, \text{internal}(r|g)) \]

denote the output state and internal state.

These states can be related to state properties via the formally defined satisfaction relation |\=|, comparable to the Holds-predicate in the Situation Calculus: \[ \text{state}(γ, t, \text{output}(r|g)) \models p \] denotes that state property p holds in trace γ at time t in the output state of role r within group g. Note that a difference between our predicate and the situation calculus one is that we refer to a trace and time point instead of a single state. Based on these statements, dynamic properties can be formulated in a formal manner in a sorted first-order predicate logic with sorts TIME or T for time points, Traces for traces and F for state formulae, using quantifiers over time and the usual first-order logical connectives such as \( \neg, \land, \lor, \rightarrow, \forall, \exists \). In trace descriptions, notations such as \[ \text{state}(γ, t, \text{output}(r|g)) \models p \] are shortened to \( \text{output}(r|g) \models p \). Consider the following dynamic property:

In any trace γ, if at any point in time t1 the agent A receives a request to perform a task, then there exists a time point t2 after t1 such that at t2 in the trace the agent A outputs a confirmation.

In formalized TTL representation it looks as follows:

\[ \forall t1 \left[ \text{state}(γ, t1, \text{input}(A)) \models \text{request(A, task)} \rightarrow \exists t2 \geq t1 \text{ state}(γ, t2, \text{output}(A)) \models \text{confirm(A, task)} \right] \]
The Temporal Trace Language can be used to specify behavioral properties at different aggregation levels. These aggregation levels can be on the level of (agent) roles, groups and the organization as a whole (see Figure 1). The lower level properties can often be modeled in simpler formats than the higher level properties. In particular, it is often possible to model the properties at the leaves of the tree in the form of directly executable properties, i.e., by direct temporal dependencies between state properties in two successive states. To model direct temporal dependencies between two state properties, not the expressive language TTL, but the simpler leads to format is used. This is an executable format that can be used to obtain a specification of a simulation model in terms of local dynamic properties (the leaves of the tree in Fig. 1). The format is defined as follows. Let $\alpha$ and $\beta$ be conjunctions of elementary state properties, and $e, t, g, h$ non-negative real numbers. (Note that $e, t, g, h$ are constants, not variables.) In the leads to language $\alpha \rightarrow_{e,t,g,h} \beta$, means:

if state property $\alpha$ holds for a certain time interval with duration $g$,
then after some delay (between $e$ and $t$) state property $\beta$ will hold for a certain time interval of length $h$.

For a precise definition of the leads to format in terms of the language TTL, see [6, Chapter 1] and [21]. A specification of dynamic properties in leads to format has as advantages that it is executable and that simulation results can be depicted graphically.

3. System Description

While conventional simulation software generates data on a given set of monitors, our extension also generates formalized traces. This Section describes our simulation system. As described above, the basis of our simulation system is a simulation software package or language, e.g., Arena, SIMULA, Swarm, etc. With such a package, a simulation model can be set up, data monitors can be set and output data will be collected. Usually this data
will be **numerical**, e.g., average waiting time, mean population size, maximum travel distance, *etc*.

This package is now extended with a software component that generates a trace consisting of formal temporally annotated state descriptions. The format of such descriptions was explained above. An example state property is

\[
\text{state}(\gamma; \text{TRACE}, T; \text{TIME}, \text{output}(\text{env}; \text{ENVIRONMENT})) \models \text{is\_at\_location}(A; \text{AGENT}, L; \text{LOCATION})
\]

saying that in trace \( \gamma \) at time \( t \) agent \( A \) is at location \( L \). A simulation trace will then consist of a list of such state properties where the variables have been instantiated. An example of a graphic representation of such a trace is included with the case study below (in Figure 4).

These state properties are the basis of more complex properties that are used either for **specification** or **validation**.

For specification, the state properties are taken up in so-called leads-to properties which were described above. In short, a leads-to property is a temporal relation between state properties. A collection of these leads-to properties implements a specification of the **system behavior**: it describes the interactions between involved entities over time. As the focus of this paper is on validation, we do not generate specifications. In related research [6, 20, 21, 22] it is described how these formal specifications may be set up manually. The automated generation of specifications is future work, as described in the discussion.

For validation, the state properties are used in **TTL** properties that were also described above. In the case study discussed below, we describe and validate several TTL properties on basis of the generated state properties. TTL properties are also temporal relations between state properties, but more general than leads-to properties. The essential difference is that leads-to properties can be **executed** (hence specification), whereas TTL properties are not necessarily executable.

With TTL properties it is possible to validate hypotheses about the system behavior. For the case study below, for example, we validate whether all agents move or not in the simulation run. The added value of TTL properties over traditional statistics or visualization of results manifests most in hypotheses that are difficult to traditionally express. For example, in grid-based simulation, **clustering** of entities is a notorious difficult problem. Although clustering may be observed by looking at the simulation while it runs, the approach is unfeasible for batch experiments, where typically hundreds of simulation runs are made to obtain reliable statistics.

Finally, TTL properties can be expressed at different aggregation levels, i.e., agent, group or system level. It is described elsewhere [20] how the relations between the aggregation levels can be analyzed.

Figure 2 gives an overview of the simulation system. As explained above, the software system is thus implemented by 1) a conventional software package (generating
the numerical output) and 2) a leads-to software component (generating the formal output).

![Diagram of simulation system]

3.1. Generation of TTL Properties

In this Section we give more insight on how to generate and formally express hypotheses in the form of TTL properties. Note, however, that most of these techniques can also be used for the generation and formulation of leads-to properties (leads-to specifications).

The first problem we address here is how to come up with a hypothesis on possible properties of the simulation. While in some cases it is fairly clear what is happening during the run of the simulation and possible properties occur naturally, often this process is not so straightforward and interesting hypotheses are not easily available. One way to provide assistance in this task is being currently investigated by the authors, namely, the use of data mining on the simulation traces to give insight on what is happening during the simulation by discovering interesting patterns. Such patterns can give rise to hypotheses that can be expressed as TTL properties and validated. We are currently experimenting with the ACE-ilProlog software package \[5, 10\] which generates query extensions – the first-order-logic equivalents of association rules. A query extension is an existentially quantified implication of the type:

$$\exists (l_1 \land \ldots \land l_m) \rightarrow \exists (l_1 \land \ldots \land l_m \land l_{m+1} \land \ldots \land l_n), \text{ with } 1 \leq m \leq n.$$ 

While a query extension is of limited expressivity compared to TTL, it might still be possible to discover unexpected, non-trivial patterns that would give inspiration for more expressive TTL properties.

The second problem we address in this Section is on the formulation of the properties in the TTL language. Obviously the properties will originally be expressed informally, in natural language. Their reformulation in TTL is not a straightforward process which includes numerous iterations of disambiguation, refinement, etc. In order to facilitate this process we use an intermediate format called structured semi-formal representation. In this representation the temporal structure is made explicit, the relevant ontology elements
are identified, references to input and output are made explicit and if necessary the property is split into its natural subparts. For example, consider property P1 expressed informally as follows:

**P1 (Informal):**
For any component production planning produced by Department A1, progress information on the production of these components will be returned to Department A1.

P1 can be reformulated in the following way (assuming some domain ontology):

**P1 (Semi-formal):**
At any point in time $t$
If at the output of Department A1 a component production planning P1 is generated, Then at a later time point $t' > t$ a progress report R1 is received at the input of Department A1.

The semi-formal representation is then used to obtain the **formal representation**. In Table 1 examples are given of how some elements of the property can be reformulated from semi-formal to formal representation:

<table>
<thead>
<tr>
<th>semi-formal</th>
<th>formal</th>
</tr>
</thead>
<tbody>
<tr>
<td>at any point in time $t$</td>
<td>$\forall t$</td>
</tr>
<tr>
<td>If (antecedent) then (consequent)</td>
<td>$[(\text{antecedent}) \Rightarrow (\text{consequent})]$</td>
</tr>
<tr>
<td>at a later point in time $t'$</td>
<td>$\exists t' \geq t$</td>
</tr>
<tr>
<td>at the output of Department A1 a component production planning P1 is generated</td>
<td>$\text{state}(\gamma, t', \text{output}(A1)) \models \text{proposed_planning}(P1)$</td>
</tr>
<tr>
<td>a progress report R1 is received at the input of Department A1</td>
<td>$\text{state}(\gamma, t', \text{input}(A1)) \models \text{progress_report}(R1)$</td>
</tr>
</tbody>
</table>

The above mentioned property P1 can now be reformulated in formal representation as follows:

**P1 (Formal):**
$\forall t \left[ \text{state}(\gamma, t', \text{output}(A1)) \models \text{proposed\_planning}(P1) \Rightarrow \text{state}(\gamma, t', \text{input}(A1)) \models \text{progress\_report}(R1) \right]$ 

For more examples of reformulation from informal to formal representation of TTL properties the reader is referred to Section 6.2 where the informal properties are denoted by IFP, the semi-formal properties by SFP and the formal properties by FP.

It should be noted that, even when all the steps of the procedure are followed faithfully, no guarantees can be given on the correctness of the formal property with respect to its informal counterpart and/or the intended hypothesis. Subtle differences in the formulation of the formal property might result in validating a different hypothesis.
from the one intended. Such mistakes are not straightforward to detect. Recent developments the related field of requirements engineering for software design also subscribes this problem and researches it as such.

There are some practical ways to approach this problem (not solve it!); we have used these pragmatic heuristics ourselves successfully while working with the framework. Firstly, we may construct a set of simple artificial traces that are guaranteed to (not) satisfy the intended property and can be used for testing - this can help in many cases although for more complex properties it might not be straightforward to generate such traces. Secondly, it might help to reformulate the property in a variety of (seemingly) equivalent versions. If they indeed express the same hypothesis then the validation result should be the same. Any deviation is an indicator of a difference in the meaning which requires further analysis including for example splitting the property to simpler sub-properties which should then be analyzed as well as the way they connect in the complex property. Finally, in previous formalizations that are effectively used for modeling real-world cases (for example, biological modeling of the human heart and e-coli bacteria, incident management, logistic modeling), the modeler (who knows about the logic best) always closely cooperated with a domain expert (who understands the properties best). The social process involved with formalizing the properties has lead to useful insights for both parties involved, and as such to best possible translations.

4. Case-study: Artificial Societies

We present an artificial environment model called VUSCAPE. The experiments presented in this paper were performed with the simulation framework JAWAS [8]. Using this framework we implemented the VUSCAPE environment and agents populating this environment.

4.1. Simulation Framework: JAWAS

The JAWAS simulation platform is comparable with existing social simulation software such as Ascape [27], Repast [9] and Swarm [11]. All settings in JAWAS can be specified either in configuration files or via command line arguments. This enables the user to automate experiments, which substantially speeds up the time needed for, for example, investigating effects of varying experimental parameters (often requiring a large number of runs). Data are automatically saved at specified locations, enabling detailed experimental logging. Thorough statistically based experimental research on complex systems that depend on numerous parameters requires a large number of runs. Facilitating this is one of the main design objectives of JAWAS. It is very easy to add, replace or delete code when changes or extensions to the model need to be implemented. No direct connection is necessary between the program and the graphical user interface. Initial exploration can use the graphical user interface, and for automated experimentation, the system can be run solely with configuration files or from the command line.
4.2. Environment model: VUSCAPE

The artificial world used in this paper is called VUSCAPE [7, 8], which is based on SUGARSCAPE [13]. For the purpose of the study described in this paper, we altered the physical and biological laws of SUGARSCAPE world in a number of ways. The adaptations extend the SUGARSCAPE domain in an interesting generic way, opening up the possibility of investigating SUGARSCAPE worlds in wider perspectives. The VUSCAPE model was investigated in the JAWAS software environment.

4.3. Task Environment

The task of each agent in the environment is the gathering of as many resources as possible during the course of its lifetime. This task was interfaced to the agent collective by using the value of the resources gathered by an agent, where gathered value translates into fitness rewards. In our system, fitness was used as a metaphor of energy: performing actions costs fitness units. Furthermore, fitness also played its conventional role in survivor selection: if an agent’s fitness reaches zero, it dies.

The environment requires the agents to cooperate in order to survive. Cooperative behavior of agents is stimulated by means of setting a maximum amount of sugar that an agent can eat on its own. It thus puts a restriction on sugar consumption. This maximum amount of sugar can be set by the cooperation threshold (CT) parameter. An agent alone can only harvest small amounts of sugar (less than CT sugar units) and it takes at least two agents to consume a large pile (more than CT sugar units). If two or more agents are at such a large pile, they share the sugar equally.

4.4. Agents

Our agents were based on the classical SUGARSCAPE design, adopting most of the SUGARSCAPE features [13]. These features include metabolism, gender, child bearing, death, vision, allow sex and replacement. An agent was able to detect agents and resources for a number of grid-cells determined by a sight property. Specifically, an agent was able to detect the number of agents and the resources in all grid-cells surrounding its current position for a distance (number of cells) given by sight.

During each simulation iteration each agent was able to move for a number of grid-cells in any position given by the value set for its move property. Both the sight and move properties were initially set to one grid-cell. Figure 3 shows the control loop of the agent.

Of the extensions that we made to the original SUGARSCAPE model, cooperation and communication are the most important. We explain these extensions here. Note that cooperation actually concerns the environment rather than the agent and therefore it was already included above in the explanation of the task environment. Since cooperation is also of consideration for the agent, we explain here how the agent is affected by it.

Cooperation. Each agent has at its disposal a maximum amount of sugar that it can harvest on its own. As mentioned previously, this amount is called the cooperation
threshold. If an agent is at a location at which the amount of sugar is over this threshold, it needs other agents to harvest the sugar. If there are more agents at such a location, these agents harvest the sugar together and the sugar is evenly distributed over these agents. In the empirical investigations described below, the cooperation threshold is the same for all agents.

Communication. Agents are endowed with talk and listen capabilities. These capabilities are implemented by probabilities that the agent will talk or listen, respectively. Per iteration it is decided whether the agent talks and/or listens or not. The talk feature determines whether the agent performs a communicative action itself, namely informing other agents of: 1) the amount of sugar that is on its location, and 2) the coordinates of its location. The listen feature is used in the observation and decision making processes of the agent. By listening, the agent receives information from other agents about amounts of sugar at the locations of those agents.

5. Experiments

This Section presents the methodology of the experiments and the obtained results.
5.1. Methodology

The aim of the empirical study described here is to investigate the response of the agent population (in terms of communication skills) to an increased need for cooperation. As described above, agents in VUSCAPE have communicative capabilities as they can multicast local information about the amount of sugar at their location and receive global information about amounts of sugar at other locations. The need for cooperation is imposed as such that agents cannot consume large amounts of sugar on their own, but rather need other agents to consume such large quantities together. Next, we describe our definitions of communication and cooperation as implemented in the experiments in more details.

5.2. Experimental Setup

Our experimental setup considers worlds with 10 x 10 locations. We conducted 5 experiment series between which we varied the number of agents {10, 20, 30, 40, 50}. For each series, we performed 5 experimental runs with communication and 5 runs without communication. In all runs, the cooperation threshold was set to 3. In total we have thus generated 50 traces. Agents and resources are initially placed on a random location. For the other experimental parameters, we have taken the default values as used elsewhere [7, 8].

Multicast messages from agents travel only over the axes and are not heard in the whole world. The decision for this is based on the fact that our agents can only move horizontally or vertically but not diagonally.

![Part of the generated VUSCAPE formal trace](image-url)
5.3. Results

Figure 4 shows part of the obtained trace from one of the experiment runs. The left side of the Figure shows a selection of the atoms that occur during the simulation. The right side shows a time-line where a dark box indicates the time intervals when an atom is true. Experiments on emergent communication and cooperation with the aim of gaining more insight into this phenomenon have been carried out and described more extensively elsewhere [7, 8].

6. Analysis

This Section presents the analysis results of our experiment series. We explain the ontology with which we formulate our properties, we present several selected properties and the validation results for these properties.

6.1. Ontology

The traces generated by the leads-to plug-in and the properties discussed in our experiments are specified using a language based on order-sorted predicate logic. Table 2 gives the sorts defined in this language with a short description.

<table>
<thead>
<tr>
<th>Sorts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE</td>
<td>the set of possible traces</td>
</tr>
<tr>
<td>AGENT</td>
<td>the set of agents</td>
</tr>
<tr>
<td>LOCATION</td>
<td>the set of locations</td>
</tr>
<tr>
<td>SUGAR</td>
<td>the set of possible sugar amounts</td>
</tr>
<tr>
<td>TIME</td>
<td>the set of possible time points</td>
</tr>
</tbody>
</table>

Furthermore there are a number of relations defined as follows:

\[
\text{is\_at\_location}: \text{AGENT} \times \text{LOCATION}, \\
\text{sugar\_at}: \text{LOCATION} \times \text{SUGAR}, \\
\text{says}: \text{AGENT} \times \text{LOCATION} \times \text{SUGAR}.
\]

Here the term \(\text{is\_at\_location}(A, L)\) would mean that agent \(A\) is at location \(L\). The term \(\text{sugar\_at}(L, S)\) means that at location \(L\) there is amount of sugar \(S\). The term \(\text{says}(A, L, S)\) means that agent \(A\) communicates that at location \(L\) there is amount of sugar \(S\).

In the next subsection we define the properties we consider in our experiments. In Section 6.3 we describe present the results.

6.2. Properties

We concentrated on a number of properties which are shortly discussed in the following paragraphs. We show the three representations of the properties (as discussed in Section
The first property is rather simple and can be expressed informally as:

IFP 1: Agents move.

After reformulation, we can express the property in semi-formal representation as follows:

SFP 1: For traces, all agents and all locations,

If an agent is now at location L1,
Then at some later time point it is at another location L2.

The formal specification of the property looks as follows:

FP 1: \( \forall \gamma: \text{TRACE}, \forall A: \text{AGENT}, \forall L_1: \text{LOCATION}, \forall t_1: \text{TIME}:
\)

\[
\text{state}(\gamma, t_1, \text{output(env)}) \models \text{is_at_location}(A, L_1) \Rightarrow
\exists L_2: \text{LOCATION}, \exists t_2: \text{TIME}:
\text{state}(\gamma, t_2, \text{output(env)}) \models \text{is_at_location}(A, L_2) \&
\]
\[\% t_2 \neq t_1 \&
L_1 \neq L_2\]

Here the sign \( \models \) indicates that the term on the right side holds in the state specified on the left side of the sign. Therefore the expression \( \text{state}(\gamma, t_1, \text{output(env)}) \models \text{is_at_location}(A, C_1) \) can be read as: “for trace \( \gamma \) at time \( t_1 \) it is true in the environment that agent \( A \) is at location \( C_1 \)”.

Property FP 1 checks whether all agents move at some point in time. If it is true, however, this does not mean that agents move all the time, nor that they move more than once during the simulation. We formulated the following property in order to get more information on how fast agents move:

IFP 1a: Agents move on within 10 time points.

The same property can be expressed in semi-formal representation as follows:

SFP 1a: For all traces, agents, locations and time points,

If an agent is at the same location at time point \( t \) and at \( t+10 \)
Then there exists a time point between \( t \) and \( t+10 \) such that the agent is at a different location.

The property can further be reformulated to a formal representation in the following way:
Formal Analysis of Dynamics of Simulated Agent Societies

FP 1a: $\forall \gamma :\text{TRACE}, \forall A: \text{AGENT}, \forall L: \text{LOCATION}, \forall t: \text{TIME}:
\begin{align*}
\text{state}(\gamma, t, \text{output(env)}) & \models \text{is\_at\_location}(A, L) \land \\
\text{state}(\gamma, t+10, \text{output(env)}) & \models \text{is\_at\_location}(A, L) \\
\exists L1: \text{LOCATION}, \exists v: \text{INTEGER}:
\text{state}(\gamma, t+v, \text{output(env)}) & \models \text{is\_at\_location}(A, L1) \land \\
v < 10 \land \\
L1 & \neq L
\end{align*}$

The second property is also simple and can be expressed informally as follows:

IFP 2: All agents communicate.

When we reformulate to semi-formal representation we refine the property in this way:

SFP 2: For all traces and all agents $A$, there exists a time point $t$, sugar amount $S$ and location $L$ such that,
- at time point $t$, agent $A$ communicates location $L$ and amount $S$.

The property can further be reformulated to a formal representation in the following way:

FP 2: $\forall \gamma :\text{TRACE}, \forall A: \text{AGENT}, \exists L: \text{LOCATION}, \exists t: \text{TIME}, \exists S: \text{SUGAR}:
\begin{align*}
\text{state}(\gamma, t, \text{output}(A)) & \models \text{says}(A, L, S)
\end{align*}$

Another property we would like to validate concerns how “truthful” agents are:

IFP 3: Agents communicate sugar locations and amounts truthfully.

What we mean by “truthfully” is that the agents communicate a location only when they are actually there and communicate sugar amount that is actually present at this location at the time. In other words, agents only send information that they have observed to be true. The property looks as follows in semi-formal representation:

SFP 3: For all traces, all agents, time points, locations,
- if an agent communicates at time $t$ that there is certain amount of sugar $s$ at a location $l$,
  - then the agent is at location $l$ at time point $t$
  - and there is amount of sugar $s$ at location $l$ at the same time.

The same property in formal language looks as follows:

FP 3: $\forall \gamma :\text{TRACE}, \forall A: \text{AGENT}, \forall L: \text{LOCATION}, \forall t: \text{TIME}, \forall S: \text{SUGAR}:
\begin{align*}
\text{state}(\gamma, t, \text{output}(A)) & \models \text{says}(A, L, S) \\
\end{align*}$
A.E. Eiben, C.M. Jonker, V. Popova, M.C. Schut

state(γ, t, output(env)) |= is_at_location(A, L) &
state(γ, t, output(env)) |= sugar_at(L, S)

The last property is a little more complex and is expressed informally as follows:

IFP 4: Sugar-rich locations attract agents.

Expressed like this, the property can be ambiguous and we need to define what we mean by “attracts”. After several iterations of reformulation, the resulting semi-formal representation of the property is as follows:

SFP 4: For all traces and all locations,

If a location is food-rich and it is not inhabited by two or more agents,

Then at some later time point there will be at least two agents at that location.

Here we need to first specify what we mean by “sugar-rich”. In the VUSCAPE environment the sugar at a location is an integer number ranging from 0 to a specified maximal number. Small amounts of sugar can be eaten by a single agent but larger amounts (a specified parameter) need at least two agents at the location who will divide the present amount in half. Here we define “sugar-rich” as containing more than two units of sugar:

\[
sugar\_rich(γ, t, l) =
\exists s: SUGAR:
\] state(γ, t, output(env)) |= sugar_at(l, s) &
\[s > 2\]

We also need to specify the criterion for “attracting agents”. We say that a location attracts agents if it is inhabited by at most one agent and at a later time there are at least two agents present at the location. Now, “at most one inhabitant” can be refined as follows: if there are two agents at the location at the same time then they are the same agent. Expressed formally this looks as:

\[
at\_most\_one\_inhabitant(γ, t, l) =
\forall A1, A2: AGENT,
\] state(γ, t, output(env)) |= is_at_location(A1, l) &
\] state(γ, t, output(env)) |= is_at_location(A2, l)
\[⇒ A1 = A2\]

Using the above definitions we can now formulate the second property:

FP 4: ∀γ: TRACE, ∀L: LOCATION, ∀t1: TIME:
sugar_rich(γ, t1, L) &
at_most_one_inhabitant(γ, t1, L) ⇒
∃ A1, A2: AGENT, ∃ t2: TIME:
state(γ, t2, output(env)) |= is_at_location(A1, L) &
state(γ, t2, output(env)) |= is_at_location(A2, L) &
t2 > t1 &
A1 ≠ A2

6.3. Analysis Results

The above discussed properties are formulated over all traces, however, in our experiments we consider a limited number of traces (namely 50) in the following experimental setting. We considered worlds with 100 locations (10 by 10) and varied number of agents {10, 20, 30, 40, 50}. In each series, 5 traces were generated with communication (the possibility for communication was switched on in VUSCAPE) and 5 without communication (the option was switched off). Each property except FP 3 was checked on all the traces. FP 3 was only checked on traces with communication since it is trivially true for the traces without communication (the antecedent is always false). The results for all properties are given in the following paragraphs.

The first property, FP 1, was found to be true for all considered traces, which was expected since the agents have to move in order to find food and therefore to survive. FP 1a was also found to be true which means that the agents move but also that they always stay at one location less than 10 time points.

The second property, FP 2, was found to be true on the traces generated with communication and false for the traces generated without communication. This was not surprising but the property was used to double-check that indeed all (not only some) agents in the traces with communication send information at least once during the trace.

The third property, however, was found to be false for all traces (with communication). This means that for each trace at least one agent communicates information that it does not observe. In order to investigate the results further we split the property in the following way:

FP 3a: ∀γ:TRACE, ∀A:AGENT, ∀L:LOCATION, ∀t:TIME, ∀S:Sugar:
state(γ, t, output(A)) |= says(A, L, S) ⇒
state(γ, t, output(env)) |= is_at_location(A, L)

FP 3b: ∀γ:TRACE, ∀A:AGENT, ∀L:LOCATION, ∀t:TIME, ∀S:Sugar:
state(γ, t, output(A)) |= says(A, L, S) ⇒
state(γ, t, output(env)) |= sugar_at(L, S)

FP 3a states that if an agent communicates information about a location then he is at that location. FP 3b states that if an agent communicates information about sugar amount
at a location then at the same time there is indeed such amount of sugar at this location. Both properties were found to be false for each trace as well.

The last property, FP 4, was also found to be false for all traces, which means that not always sugar-rich locations attract more than one visitor at the same time.

For properties FP 3, FP 3a, FP 3b and FP 4 the software we use provided us with instantiations for which each property fails. This information can be used to pinpoint for which agent, location, time point, etc. the property does not hold. Such details can help in finding the reasons why the property fails, however, in this process other tools will be used too and often we would need to go back to the simulation software and perform additional experiments under different conditions. When a hypothesis is created on what is actually happening during the trace, it can be formulated as a new TTL property which in turn will be checked on the simulation traces.

7. Discussion

We introduced an extension component for simulation software capable of generating formal output. Our proposed system includes a simulation software package or language as a starting point, which is extended with a part that generates formal traces consisting of formal state properties. While conventional simulation software generally outputs numerical data, our simulation system also outputs formal data.

We performed a case study in the area of social simulation. As such, our simulation system included an existing testbed for researching artificial societies and generated formal traces containing information about the locations of agents, resource locations and communication between the agents. Based on these properties, we were able to validate hypotheses, e.g., whether agents moved frequently from one location to another during simulation runs and whether locations with larger amount of sugar attract more agents.

The example was kept simplistic on purpose, since this paper does not have the aim to deliver new results in the discipline of social simulation. Instead, the case study was used to illustrate our simulation system.

Our approach relies on a number of assumptions listed in the introduction. For some of them (the assumptions on the translation from an informal to a formal property) were discussed in Section 3.1 and some heuristics were given to guide the translation process. The rest of the assumptions point to possible future research direction. For example more analysis is necessary on the formal languages for expressing traces and properties: what requirements they need to satisfy, what types of languages can be used, etc. Furthermore guidelines can be formulated on the generation of the traces, e.g. how to “calibrate” the expressivity and the time frame of the trace so that the relevant behavior is captured while the size of the trace is still kept within the acceptable boundaries.

More concrete future work directions that we consider are as follows. Firstly, we plan to investigate the interfacing between the simulation language and our formal language. The aim of this research path is to allow simulators to specify themselves what information is collected in the formal traces. Secondly, we plan develop a set of learning algorithms with which system behavior specifications can be generated automatically.
Formal Analysis of Dynamics of Simulated Agent Societies

Technically, from a set of instantiated state properties, a set of (generic) leads-to or TTL properties will be generated. Finally, our overall goal is to develop a methodology to analyze and design complex interaction-based systems that integrates empirical and formal research approaches.

Acknowledgements

We would like to thank Jan Treur for the inspiring discussions held on the topic of this paper. Also, we thank Geoff Nitschke, Nico Vink, Volker Nannen, and Robert Griffioen for their contributions to the JAWAS platform and the stimulating discussions.

References