OPERAS: A Framework for the Formal Modelling of Multi-Agent Systems and Its Application to Swarm-Based Systems

Ioanna Stamatopoulou[1], Petros Kefalas[2], and Marian Gheorghe[3]

[1] South-East European Research Centre, Thessaloniki, Greece
istamatopoulou@seerc.org
[2] Department of Computer Science, CITY College, Thessaloniki, Greece
kefalas@city.academic.gr
[3] Department of Computer Science, University of Sheffield, UK
M.Gheorghe@dcs.shef.ac.uk

Abstract. Swarm-based systems are a class of multi-agent systems (MAS) of particular interest because they exhibit emergent behaviour through self-organisation. They are biology-inspired but find themselves applicable to a wide range of domains, with some of them characterised as mission critical. It is therefore implied that the use of a formal framework and methods would facilitate modelling of a MAS in such a way that the final product is fully tested and safety properties are verified. One way to achieve this is by defining a new formalism to specify MAS, something which could precisely fit the purpose but requires significant period to formally prove the validation power of the method. The alternative is to use existing formal methods thus exploiting their legacy. In this paper, we follow the latter approach. We present *OPERAS*, an open framework that facilitates formal modelling of MAS through employing existing formal methods. We describe how a particular instance of this framework, namely $OPERAS_{XC}$, could integrate the most prominent characteristics of finite state machines and biological computation systems, such as X-machines and P Systems respectively. We demonstrate how the resulting method can be used to formally model a swarm system and discuss the flexibility and advantages of this approach.

1 Introduction

Despite the counter arguments which justifiably raise concerns about formal methods, there is still a strong belief by the academic community that the development of mission critical systems demands the use of such methods for modelling, verification and testing. Opposition puts forward a significant drawback; the more complex a system is, the more difficult the modelling process turns out to be and, in consequence, the less easy it is to ensure correctness at the modelling and implementation level. Correctness implies that all desired safety properties are verified at the end of the modelling phase and that an appropriate testing technique is applied to prove that the implementation has been built in accordance to the verified model. The formal methods community has made significant progress towards the development of correct systems.

On the other hand, multi-agent systems (MAS) are complex software systems by default. Especially when the agent society grows, interaction and communication increase within a complex structure which involves variety of knowledge, abilities, roles and tasks. Nature seems to have found ways to deal with complex structures quite effectively. Consider, for example, biological systems from the smallest living elements, the cells, and how they form tissues in organisms to entire ecosystems and how they evolve [1]. There is growing interest in investigating ways of specifying such systems. The intention is to create software that mimics the behaviour of their biological counterparts. Examples of biological systems of interest also include swarm-based systems, such as social insect colonies.

The promising feature is that these systems can be directly mapped to MAS by considering each entity as an agent, with its own behavioural rules, knowledge, decision making mechanisms and means of communication with the other entities and with the environment. The overall system's behaviour is merely the result of the agents' individual actions, the interactions among them and between them and the environment. This also points to the issue of self-organisation and how collective behavioural patterns emerge as a consequence of individuals' local interactions in the lack of knowledge of the entire environment or global control.

An additional key modelling aspect of swarm-based systems is their dynamic nature and how their structure is constantly mutated. By *structure* we imply: the changing number of agents, and either their physical placement in the environment or, more generally, the structure that is dictated by the communication channels among them. Other classes of MAS also exhibit reorganisational change, characterised as behavioural or structural change [2].

Existing wide-spread formal methods fail to provide the appropriate features in order to model such dynamic system organisation —most of them assume a fixed, static structure that is not realistic (e.g. cellular and communicating automata), since communication between two agents may need to be established or ceased at any point and also new agents may appear in the system while existing ones may be removed. It is fairly recently that the issue of structural change is attempted to be in essence dealt with, and this poses a kind of dilemma: should a completely new formal notation be devised or should existing ones be used and possibly be improved? Both approaches have complementary advantages; a new formal method will directly tackle the problem of modelling of change but existing ones will carry the legacy of formal testing and verification.

In this paper, we deal with the latter approach. The next section introduces the $OPERAS$ formal definition as a framework for modelling MAS, while Section 3 presents an instance of this framework, namely $OPERAS_{XC}$, which utilises existing formal methods. A brief description of a representative case study dealing with a swarm-based system follows in Section 4 which also deals with the formal model for the case problem in question. Finally, Section 5 discusses issues arising from our attempt and concludes the paper.

2 OPERAS: Formal Modelling of MAS

2.1 Background and Related Work

In an attempt to formally model each individual agent as well as the dynamic behaviour of the overall system, a formal method should be capable of rigorously describing all the essential aspects, i.e. knowledge, behaviour, communication and dynamics. There is a number of trade-offs on the use of formal methods for MAS. To name a few: (a) the level of abstraction should be appropriate enough to lead toward the implementation of a MAS but also be appropriate enough to mathematically express specifications that can lead to formal verification and complete testing, (b) there should be accompanying toolkits which make their adoption wider by researchers and industry but at the same time the tools provided should not deviate from the theoretical framework, (c) they ought to provide means to efficiently define complex knowledge but also be able to describe control over individual agent as well as MAS states, (d) they need to be able to easily model individual agents but also to focus on the concurrency and communication among them.

In agent-oriented software engineering, several approaches using formal methods have been proposed, each one focusing on different aspects of MAS development. For example, with respect to the issue of organisation, there is a large number of approaches employing formal methods in modelling MAS and focusing on organisational reconfiguration [3, 4, 5], specificational adaptation at run time [6] and formal methodologies to engineer organisation-based MAS [7]. Other efforts have been directed toward moving to the implementation of a MAS through refinement of the specification and developing proof theories for the architecture [8], capturing the dynamics of an agent system [9], putting emphasis on capturing and controlling the system dynamics and acting behaviour of MAS [10]. Other approaches formally specify MAS and then directly execute the specification while verifying important temporal properties [11] or guide through a prototyping process [12]. Less formal approaches, which accommodate the distinctive requirements of agents, have been proposed [13]. Additionally, there is a set of general principles for capturing the organisational structure of MAS [14] which are however linked more to implementation [15] rather than formal modelling.

On the other hand, from a purely software engineering view, a plethora of formal methods are provided (Z, VDM, FSM, Petri-Nets, CCS, CSP), with none of them alone satisfying all the above mentioned criteria for MAS, but with a rich legacy on specification, semantics, testing and verification. Other formal methods, such as π-calculus, mobile ambients and P Systems with mobile membranes [16, 17, 18, 19], successfully deal with the dynamic nature of systems and concurrency of processes but lack intuitiveness when it comes to the modelling of an individual agent (lack of primitives and more complex data structures). Lately, new computation approaches as well as programming paradigms inspired by biological processes in living cells, introduce concurrency as well as neatly tackle the dynamic structure of multi-component systems (P Systems, Brane Calculus, Gamma, Cham, MGS) [20, 21, 22]. An interesting comparison of various formal methods for the verification of emergent behaviours in swarm-based systems is reported in [23], where an asteroid exploration scenario by autonomous spacecrafts is considered. We will use the same scenario in order to benchmark our approach.

2.2 OPERAS Definition

Our aim is to define a framework in which we can use existing formal methods to model classes of MAS where self-organisation and emergent behaviour is achieved through a number of changes in their structure. As said in the previous section, none of the existing formal methods qualify to deal equally well with individual agent modelling as well as dynamics of the system. We believe that the problem will be solved by combining formal methods. But for doing so, we should somehow distinguish between the modelling of the individual agents (behaviour) and the rules that govern the change in the structure of the collective MAS (structure mutation). This distinction, which would greatly assist the modeller by breaking down the work into two separate and independent activities, may be achieved by considering that each agent is wrapped by a separate mechanism: a structural mutator. Extending an agent with a kind of a wrapper is not a novel idea in MAS engineering though it has been primarily used for communication purposes and not in the context of formal specification. In this case, we refer to a structural mutator as the independent part of the agent that is responsible for checking an agent's internal computation state and its local environment in order to determine whether a structural change in the system has to take place, might that be the addition/removal of communication channels or other agents.

In general terms, when modelling a MAS, one should specify a number of agents, the environment in which they operate, the stimuli provided from the environment as percepts to the agents, the agents abilities and roles, the agents grouping and organisation and communication between them. A *Multi-Agent System* model in its general form, as it is perceived from a formal modelling perspective can be defined by the tuple (O, P, E, R, A, S) containing:

- a set of reconfiguration rules, O, that define how the system structure evolves by applying appropriate reconfiguration operators;
- a set of percepts, P, for the agents;
- the environment's model / initial configuration, E;
- a relation, R, that defines the existing communication channels;
- a set of participating agents, A, and
- a set of definitions of types of agents, S, that may be present in the system.

The definition is general enough not to restrict any organisational structure that might be considered for the implementation of a MAS. In addition, the definition could be further extended to include protocols or other features of MAS that a modeller would wish to formally specify. For now, *OPERAS* fits our purpose, that of modelling swarm-based systems. More particularly:

- the rules in O are of the form *condition* $\Rightarrow$ *action* where *condition* refers to the computational state of agents and *action* involves the application of one or more of the operators that create / remove a communication channel between agents or introduce / remove an agent into / from the system;
- P is the distributed union of the sets of percepts of all participating agents;
- $R : A \times A$ with $(A_i, A_j) \in R$, $A_i, A_j \in A$ meaning that agent A_i may send messages to agent A_j;
- $A = \{A_1, \ldots A_n\}$ where A_i is a particular agent defined in terms of its individual behaviour and its local mechanism for structure mutation;
- $S_k = (Behaviour_k, StructureMutator_k) \in S$, $k \in Types$ where $Types$ is the set of identifiers of the types of agents, $Behaviour_k$ is the part of the agent that deals with its individual behaviour and $StructureMutator_k$ is the local mechanism for structure reconfiguration; each participating agent A_i of type k in A is a particular instance of a type of agent: $A_i = (Beh_k, StrMut_k)_i$.

2.3 OPERAS as an Open Framework

The general underlying idea is that an agent formal model consists of two parts, its behaviour and its structural mutator. The behaviour of an agent can be modelled by a formal method with its computation being driven by percepts from the environment. The structural mutator can be modelled by a set of reconfiguration rules which given the computation states of agents can change the structure of the system. The MAS structure is determined through the relation that defines the communication between the agents. The set of participating agents are instances of agent types that may participate in the system. This deals with the fact that an agent may be present at one instance of the system but disappear at another or that a new agent or a new role comes into play during the evolution of the MAS. This assumes that all agent types and roles that may participate in the system should be known in advance.

There are still some open issues which, however, make the *OPERAS* approach a framework rather than a formal method. These are: (i) Which formal method may we use in order to model the agents' behaviour? (ii) Which formal method may we use in order to model the structural mutator? (iii) Could the methods in (i) and (ii) be different? (iv) Should the formal method used in (i), for modelling behaviour, provide features for communication directly or indirectly (implicitly through percepts from the environment) among agents' behaviours? (v) Should the formal method used in (ii), for modelling structure mutation, provide features for communication directly or indirectly (through the environment)among agents' structure mutators? (vi) Which method chosen from (i) or from (ii) drives the computation of the resulting system? There is no unique answer to these questions but the choice of formal methods which are considered suitable to model either behaviour or structure mutation may affect the final model developed.

It is therefore implied that there are several options which could instantiate *OPERAS* into concrete modelling methods. Regarding the modelling of each type of agent S_k, there are more than one options to choose from in order to specify its behavioural part and the same applies for its structure mutation mechanism. We have long experimented with two formal methods, which are X-machines with its communicating counterpart and Population P Systems (PPS) with active cells. We use X-machines because they demonstrated considerable power in modelling reactive systems and most importantly they are accompanied by two distinctive features: a complete testing strategy and a well-defined model checking methodology. We chose Population P systems because of their theoretically sound way to model computation taking place inside a membrane-like dynamic system. Ad hoc integration of these two methods [24, 25, 26] gave us some preliminary results which led us to the current combined approach we take for *OPERAS*. It is interesting to notice that none of the two formal methods by itself could successfully (or at least intuitively) model a MAS [24, 25]. This is also true, although with better results, if we use only PPSs under the OPERAS framework ($OPERAS_{CC}$) [27]. The problem still exists for other formal methods too, which means the current framework gives the opportunity to combine those methods that may be best suited to either of the two modelling tasks. In the following, we present an instance of *OPERAS*, named $OPERAS_{XC}$, that uses Communicating X-machines and features from PPSs.

3 OPERAS$_{XC}$

3.1 Modelling Behaviour

X-machines (XM), a state-based formal method introduced by Eilenberg [28], are considered suitable for the formal specification of a system's components. Stream X-machines, in particular, were found to be well-suited for the modelling of reactive systems. Since then, valuable findings using the X-machines as a formal notation for specification, communication, verification and testing purposes have been reported [29, 30, 31]. An X-machine model consists of a number of states and also has a memory, which accommodates mathematically defined data structures. The transitions between states are labelled by functions. More formally, a stream X-machine is defined as the 8-tuple $(\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ where:

- Σ and Γ are the input and output alphabets respectively;
- Q is the finite set of states;
- M is the (possibly) infinite set called memory;
- Φ is a set of partial functions φ that map an input and a memory state to an output and a possibly different memory state, $\varphi : \Sigma \times M \rightarrow \Gamma \times M$;
- F is the next state partial function, $F : Q \times \Phi \rightarrow Q$, which given a state and a function from the type Φ determines the next state. F is often described as a state transition diagram;
- q_0 and m_0 are the initial state and initial memory respectively.

X-machines can be thought to apply in similar cases where StateCharts and other similar notations do. In principle, X-machines are considered a generalisation of models written in such formalisms.

In addition to having stand-alone X-Machine models, communication is feasible by redirecting the output of one machine's function to become input to a function of another machine. The structure of a *Communicating X-machines* (CXM) system is defined as the graph whose nodes are the components and edges are the communication channels among them (Fig. 1). A formal definition of CXMs can be found in [24].

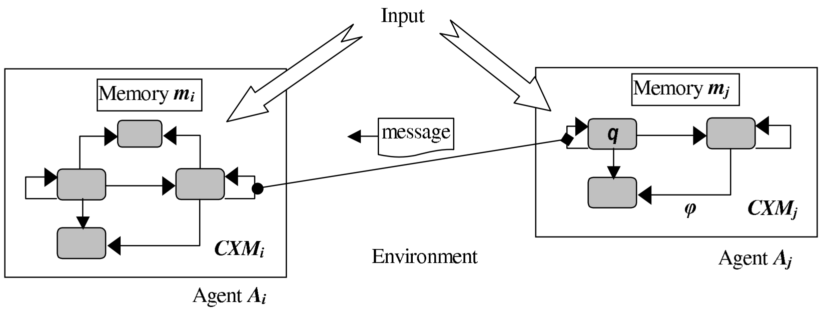

Fig. 1. An abstract system consisting of two CXM components. Communication is established by redirecting the output of a function (⧫ symbol) to another machine's function which takes it as input (● symbol).

CXMs provide a straightforward way for dealing with an agent's behaviour, however, the structure of a communicating system must be known beforehand and fixed throughout the computation.

3.2 Modelling Structure Mutation

A *Population P System* [32] is a collection of different types of cells evolving according to specific rules and capable of exchanging biological / chemical substances with their neighbouring cells (Fig. 2). More formally, a PPS is defined as a construct $\mathcal{P} = (V, K, \gamma, \alpha, w_E, C_1, C_2, \ldots, C_n, R)$ where:

- V is a finite alphabet of symbols called objects;
- K is a finite alphabet of symbols, which define different types of cells;
- $\gamma = (\{1, 2, \ldots n\}, A)$, with $A \subseteq \{\{i, j\} \mid 1 \leq i \neq j \leq n\}$, is a finite undirected graph;
- α is a finite set of bond-making rules;
- $w_E \in V^*$ is a finite multi-set of objects initially assigned to the environment;
- $C_i = (w_i, t_i)$, for each $1 \leq i \leq n$, with $w_i \in V^*$ a finite multi-set of objects, and $t_i \in K$ the type of cell i;
- R is a finite set of rules dealing with object transformation, object communication, cell differentiation, cell division and cell death.

Transformation rules replace an object within a cell. *Communication rules* allow the exchange of objects between neighbouring cells, or a cell and the environment, according to the cell type and the existing bonds among the cells. *Cell*

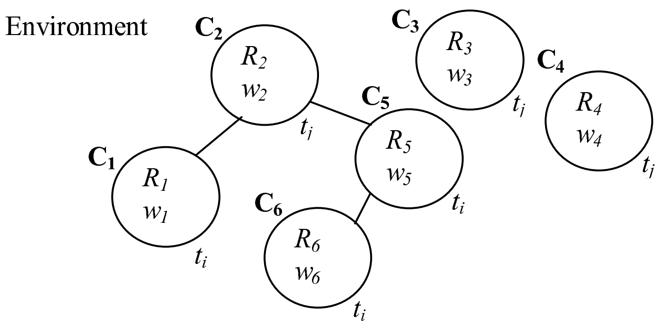

Fig. 2. An abstract example of a Population P System; C_i: cells, R_i: sets of rules related to cells; w_i: multi-sets of objects associated to the cells.

differentiation rules change a cell, transforming it into a cell of a new type. *Cell division rules* divide a cell into two cells. *Cell death rules* cause the removal of a cell from the system.

At each computation cycle, all rules regarding the transformation and communication of objects that may be applied in a cell are applied. Additionally, one out of the applicable cell differentiation, division or death rules, non-deterministically chosen, is also applied in each cell. When computation in all cells has finished, the graph is decomposed and restructured according to the specified bond-making rules in α that define the conditions under which two cells are able to communicate.

PPS provide a straightforward way for dealing with the change of a system's structure, however, the rules specifying the behaviour of the individual cells (agents) are more commonly of the simple form of rewrite rules which are not sufficient for describing the behaviour of the respective agent.

3.3 Definition of OPERAS$_{XC}$

We may now move on to a more formal $OPERAS_{XC}$ definition that uses both a CXM (indicator subscript X) and PPS-cell-inspired construct (indicator subscript C) for specifying each of the agents. An abstract example of an $OPERAS_{XC}$ model consisting of two agents is depicted in Fig. 3.

For the following, we consider that the computation state of a CXM describing the behaviour of an agent is a 3-tuple $Q \times M \times \Phi$ that represents the state the XM is in (q_i), its current memory (m_i) and the last function that has been applied (φ_i).

A MAS in $OPERAS_{XC}$ is defined as the tuple (O, P, E, R, A, S) where:

- The rules in O are of the form *condition* $\Rightarrow$ *action* where *condition* is a conjunction of (q, m, φ) and *action* involves the application of one or more of the operators attachment **ATT** and detachment **DET**, which reconfigure the communication channels among existing CXMs and generation **GEN** and destruction **DES**, which generate or destroy an agent in/from the system. Additional

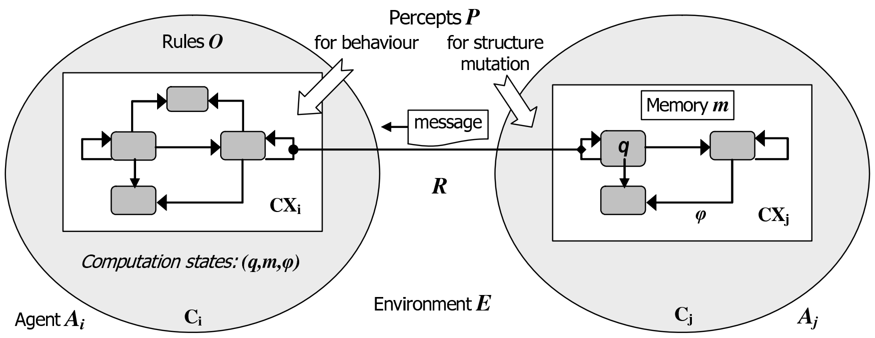

Fig. 3. An abstract example of a $OPERAS_{XC}$ consisting of two agents

communication rules also exist, as in PPS, so that there is indirect communication (through the environment) between the structural mutators (cells);

– $P = P_B \cup P_{SM}$ is the set of percepts of all participating agents, where $P_B = \Sigma_1 \cup \ldots \cup \Sigma_t$ is the set of inputs perceived by the XM model of the behaviour (subscript B) and $P_{SM} = (Q_1 \times M_1 \times \Phi_1) \cup \ldots \cup (Q_1 \times M_t \times \Phi_t)$ is the set of objects (alphabet) of the PPS mechanism that captures structure mutation (subscript SM), t being the number of types of agents;
– $E = \{(q, m, \varphi)_i | 1 \leq i \leq n, q \in Q_i, m \in M_i, \varphi \in \Phi_i\}$ holding information about the initial computation states of all the participating agents;
– $R : CXM \times CXM$ (CXM: the set of CXMs that model agent behaviour);
– $A = \{A_1, \ldots, A_n\}$ where $A_i = (CXM_k, C_k)_i$ is a particular agent of type k defined in terms of its individual behaviour (CXM_k) and its local structural mutator cell for controlling reconfiguration (C_k). The structural mutator cell is of the form $C_k = (w_i, o_k)$ where w_i is the multi-set of objects it contains and $o_k \subset O$ is the set of rules that correspond to the particular type of agent, k;
– $S = \{(XT_k, C_k) | \forall k \in Type\}$, where XT_k is an XM *type* (no initial state and memory).

The above mentioned operators attachment **ATT** and detachment **DET** have the same effect as the bond-making rules of a PPS, while the operators generation **GEN** and destruction **DES**, have the same effect as cell division an cell death of a PPS respectively. Formal definitions of these operators can be found in [33].

In this model, each structural mutator cell implicitly knows the computation state (q, m, φ) of the underlying XM that models behaviour. Environmental input is directed straight to the agent's behavioural part. In each computation cycle an input triggers a function of the behaviour CXM and the updated information about the agent's current computation state is updated in the structural mutator cell. A copy of the object is placed in the environment for other agents in the local environment to have access to it. Objects from the environment representing the computation states of neighbouring agents are imported and finally, all the reconfiguration rules in O of the type of the particular cell are being checked and if necessary applied. Since the model follows the computation rules of a CXM system (triggered by the behaviour component's input, asynchronously for the different participating agents), computation of the behaviour-driven version of $OPERAS_{XC}$ is *asynchronous*. In another version of $OPERAS_{XC}$, the computation is cell-driven, and therefore *synchronous*. A detailed and more formal analysis of the two versions, however, falls outside the scope of this paper. In addition, as said previously, other instances of $OPERAS$ using these two methods, such as $OPERAS_{CC}$, $OPERAS_{XX}$ and $OPERAS_{CX}$ are possible but rather cumbersome.

4 OPERAS$_{XC}$ for a Swarm-Based System

4.1 Autonomous Spacecrafts for Asteroid Exploration

A representative example of a system which clearly possesses all the aforementioned characteristics of a dynamic MAS is the NASA Autonomous Nano-Technology Swarm (ANTS) system [23]. The NASA ANTS project aims at the development of a mission for the exploration of space asteroids with the use of different kinds of unmanned spacecrafts. Though each spacecraft can be considered as an autonomous agent, the successful exploration of an asteroid depends on the overall behaviour of the entire mission, as the latter emerges as a result of self-organisation. We chose this case study because correctness of the system has been identified as a primary requirement. Relevant work on the particular project included research on and comparison of a number of formal methods [23, 34], including CXMs.

The ANTS mission uses of three kinds of unmanned spacecrafts: leaders, L, (or rulers or coordinators), workers, W, and messengers, M (Fig. 4). The leaders are the spacecrafts that are aware of the goals of the mission and have a non-complete model of the environment. Their role is to coordinate the actions of the spacecrafts that are under their command but by no means should they be considered to be a central controlling mechanism as all spacecrafts' behaviour is autonomous. Depending on its goals, a leader creates a team consisting of a number of workers and at least one messengers. Workers and messengers are assigned to a leader upon request by (i) another leader, if they are not necessary for the fulfilment of its goals, or (ii) earth (if existing spacecrafts are not sufficient in number to cover current needs, new spacecrafts are allocated to the mission).

A worker is a spacecraft with a specialised instrument able, upon request from its leader, to take measurements from an asteroid while flying by it. It also possesses a mechanism for analysing the gathered data and sending the analysis results back to its leader in order for them to be evaluated. This in turn might update the view of the leader, i.e. its model of the environment, as well as its future goals.

The messengers, finally, are the spacecrafts that coordinate communication among workers, leaders and the control centre on earth. While each messenger is under the command of one leader, it may also assist in the communication of other leaders if its positioning allows it and conditions demand it.

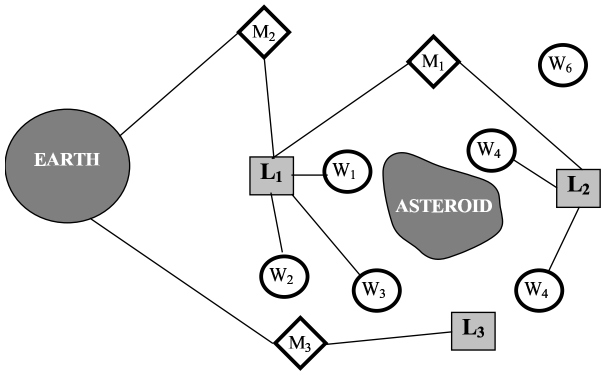

Fig. 4. An instance of the ANTS mission, L: Leader, W:Worker, M:Messenger

What applies to all types of spacecrafts is that in the case that there is a malfunctioning problem, their superiors are being notified. If the damage is irreparable they need to abort the mission while on the opposite case they may "heal" and return back to normal operation.

4.2 Leader: Formal Modelling of Behaviour in OPERAS$_{XC}$

The leader agent L can be modelled as an XM, whose state transition diagram F_L is depicted in Fig. 5. $Q_L = \{Processing, Malfunctioning, Aborting\}$ is the set of states a leader may be in. Its memory contains information about its current status (i.e. position and operational status), the IDs and statuses of the messengers and workers under its command, the analysis results up to this point, its current model of the surroundings as well as its goals: $M_L : \mathcal{S}tatus \times \mathbb{P}(\mathcal{M} \times \mathcal{S}tatus) \times \mathbb{P}(\mathcal{W} \times \mathcal{S}tatus) \times \mathcal{A}nalysisResults \times \mathcal{M}odel \times \mathcal{G}oals$ where $\mathcal{S}tatus : (Z \times Z \times Z) \times \{Q_L\}$ (Z being the set of positive integers, the 3-tuple denoting a position), $\mathbb{P}$ stands for power-set, $\mathcal{M}$ is the set of messengers, $\mathcal{W}$ is the set of workers and so forth.

The input set for the leader XM is $\Sigma_L = \{abrt, problem, remedy\} \cup (\mathcal{W} \times \mathcal{S}tatus) \cup (\mathcal{W} \times \mathcal{M}easurements) \cup (\{request, requestFromEarth, requestedFor\} \times \mathcal{I}nstrument)$, where $abrt$, $problem$, $remedy$, $request$, $requestedFor$ are constants and $\mathcal{I}nstrument$ is the set of containing the different types of installed instruments of the workers. The output set Γ_L is a set of informative messages.

Indicatively, some of the functions in the Φ_L set (functions are of the form: $function(input, memory\_tuple) = (output, memory\_tuple'))$ are:

$acceptRequestForWorker\ ((requestedFor, instr), (\_, \_, workers, \_, \_, \_)) =$
$('reassigned\ worker', (\_, \_, workers', \_, \_, \_))$
if $(w_i, (\_, \_, instr)) \in workers$
and $isWorkerNeeded(w_i) == false$
where $workers' = workers \backslash (w_i, (\_, \_, instr))$

$receiveWorker(w_i, (\_, \_, workers, \_, \_, \_)) =$
$('received\ worker', (\_, \_, workers \cup (w_i), \_, \_, \_))$

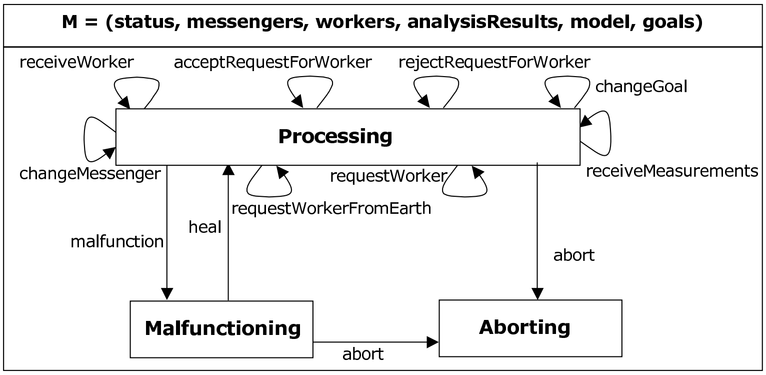

Fig. 5. State transition diagram of the Leader X-machine

As aforementioned, we used XMs for agent formal modelling because they facilitate formal verification and testing. These operations are crucial in developing mission critical systems. $\mathcal{X}m$CTL , an extension of CTL for XMs, can be used to verify models against the requirements, since it can prove that certain properties are true. Such properties are implicitly encoded in the memory structure of the XM model [30]. For example, the property "there exists a computation path in which a leader will accomplish all its goals and in all previous states the leader was employing at least one worker" is specified in $\mathcal{X}m$CTL as:

$$\mathbf{E}[\mathbf{M_x}(mem_L(3) \neq \emptyset)\ \mathbf{U}\ \mathbf{M_x}(mem_L(6) = \emptyset)]$$

where $mem_L(i)$ indicates the i-th element in the memory tuple of the leader model. Additionally, it is possible under certain well defined conditions, to produce a complete test set out of an XM model. The test set guarantees to determine the correctness of the implementation of each agent [31].

4.3 Worker: Formal Modelling of Behaviour in OPERAS$_{XC}$

The state transition diagram of the worker XM is depicted in Fig. 6. The internal states in which a worker may be are $Q_W = \{Measuring, Analysing, Malfunctioning, Aborting\}$ and its memory holds information about its current status (i.e. position, operational status and installed instrument), the identity and status of its commanding leader, the messengers which assist its communication, the target asteroid, the data received from the measurements and the results of the data analysis: $M_W : \mathcal{S}tatus \times (\mathcal{L} \times \mathcal{S}tatus) \times \mathbb{P}(\mathcal{M} \times \mathcal{S}tatus) \times \mathcal{T}arget \times \mathcal{M}easurements \times \mathcal{A}nalysis\mathcal{R}esults$.

The input set is $\Sigma_W = \{measure, analyse, send, abrt, problem, remedy\} \cup (\mathcal{L} \times \mathcal{S}tatus)$, where $abrt$, $problem$, $remedy$, $measure$, $analyse$ and $send$ are constants. The output set Γ_W is a set of informative messages.

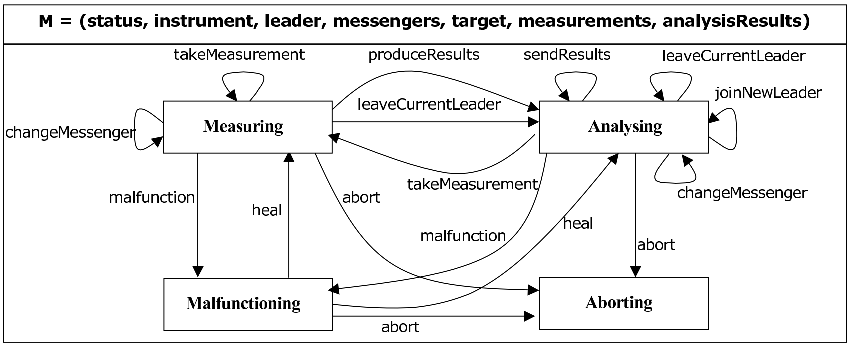

Fig. 6. State transition diagram of the Worker X-machine

Indicatively, some of the functions in the Φ_W set are:

$produceResults(analyse, (\_, \_, \_, \_, meas, analysisResults)) =$
$('analysed', (\_, \_, \_, \_, \emptyset, analysisResults')),$
where $analysisResults' = analysisMechanism(meas) :: analysisResults$

$sendResults(send, (\_, \_, \_, \_, \_, res :: analysisResults)) =$
$('sent\ results', (\_, \_, \_, \_, \_, analysisResults))$

$leaveCurrentLeader((newLeader, st), (\_, (leader, st_0), \_, \_, \_, \_)) =$
$('been\ reassigned', (\_, newLeader, \_, \_, \_, \_))$

The model of the messenger agent is similarly created.

4.4 Formal Modelling of Structure Mutation in OPERAS$_{XC}$

According to $OPERAS_{XC}$, for the definition of the given system as a dynamic MAS, we need to assume an initial configuration. To keep the size restricted for demonstrative purposes, let us consider an initial configuration that includes one leader L_1, one messenger M_1 and two workers W_1, W_2.

The set O contains the following reconfiguration rules regarding: (a) generation of a new worker when the control centre on earth decides it should join the mission, (b) the destruction (i.e. removal from the system) of any kind of agent in the case it must abort the mission, (c) the establishment of a communication channel between a leader and a newly assigned to it worker, and (d) the removal of a communication channel between a leader and a worker when the latter is being reassigned to a new leader.

More particularly O contains the following rules:

If there is a need for an additional worker and earth can allocate one than a new agent appear in system ANTS:

$(\_, \_, requestWorkerFromEarth)_{L_i} \wedge earthHasAvailableWorkers() == true$
$\Rightarrow \mathbf{GEN}(W_i, q_{0_i}, m_{0_i}, ANTS)_L$

If an agent aborts the mission then the agent is removed from system ANTS:

$(aborting, \_, \_)_{*_{this}} \Rightarrow \mathbf{DES}(*_{this}, ANTS)_*$

If a worker agent looses its contact with its leader then the communication channels between the two agents are broken:

$(\_,(\_,\_,L_i,\_,\_,\_,\_), leaveCurrentLeader)_{W_i}$
$\Rightarrow \mathbf{DET}(W_i, L_i, \mathbf{DET}(L_i, W_i, ANTS))_W$

If a worker agent is assigned with a new leader then a new communication channel is established:

$(\_,(\_,\_,newLeader,\_,\_,\_,\_), joinNewLeader)_{W_i}$
$\Rightarrow \mathbf{ATT}(W_i, newLeader, ANTS)_W$

If a leader agent is assigned with a new worker (either from another leader or from earth) then a new communication channel is established:

$(\_,(\_,\_,newWorker :: workers,\_,\_,\_), receiveWorker)_{L_i}$
$\Rightarrow \mathbf{ATT}(L_i, newWorker, ANTS)_L$

$(\_,(\_,\_,newWorker :: workers,\_,\_,\_), receiveWorkerFromEarth)_{L_i}$
$\Rightarrow \mathbf{ATT}(L_i, newWorker, ANTS)_L$

where * stands for any type of agent.

The set of percepts of all agents is:

$$P = \Sigma_L \cup \Sigma_W \cup \Sigma_M \cup (Q_L \times M_L \times \Phi_L) \cup (Q_W \times M_W \times \Phi_W) \cup (Q_M \times M_M \times \Phi_M).$$

Because all reconfiguration rules per type of agent rely only on conditions dependent on the computation state of the agent itself (and not other agents), the model needs not to communicate computation states among the different agents and there are, therefore, no additional communication rules. A direct consequence of this is that there is no need for the environment to play the role of communication mediator between the participating entities and as such no need for it to hold any computation state objects: $E = \emptyset$.

Since in the assumed initial configuration we consider to have one group of spacecrafts under the command of one leader, all agents should be in communication with all others and so:

$R = \{(L_1, W_1), (L_1, W_2), (L_1, M_1), (M_1, L_1), (M_1, W_1), (M_1, W_2), (W_1, L_1), (W_1, M_1), (W_2, L_1), (W_2, M_1)\}$

The set that contains all the agent instances becomes: $A = \{L_1, W_1, W_2, M_1)\}$ where $L_1 = (CXM_{L_1}, C_{L_1})$, $W_i = (CXM_{W_i}, C_{W_i}), 1 \leq i \leq 2$ and $M_1 = (CXM_{M_1}, C_{M_1})$.

Finally, the set S that contains the "genetic codes" for all agent types is:

$S = \{(XT_L, C_L), (XT_W, C_W), (XT_M, C_M)\}$ where L, W, M are the XMs defined previously.

5 Conclusions and Further Work

We presented *OPERAS*, a framework, with which one can formally model the behaviour and control over the internal states of an agent as well as formally describe the mutations that occur in the structure of a MAS, as two separate components. Driven by a formal methods perspective, we employed CXMs and ideas from PPSs to define $OPERAS_{XC}$, a particular instance of the framework. These gave us the opportunity to combine the advantages that XMs have in terms of modelling the behaviour of an agent, testing it and verifying its properties with the advantages that PPSs have in terms of defining the mutation of the structure of a MAS. We have experimented with modelling of various biological and biology-inspired systems. In this paper we presented the $OPERAS_{XC}$ model of a swarm-based system of a number of autonomous spacecrafts, a case which has been used by researchers for comparative study of formal methods.

We would like to continue the investigation of how *OPERAS* could employ other formal methods that might be suitable for this purpose. In the near future, we will focus on theoretical aspects of the framework, in order to demonstrate its usefulness towards developing correct agent societies (i.e. complete testing and verification). Although work on verification and testing has been done with XMs [30, 31], it is important to investigate to what extent this could be inherited in a hybrid system, like $OPERAS_{XC}$. Towards this direction, we are also currently working on various types of transformations that could prove its power for formal modelling as well as address legacy issues with respect to correctness [35]. These developments are mainly of interest to the formal method community.

On the other hand, the MAS community might be interested in how $OPERAS_{XC}$ can facilitate the implementation of agent systems. Towards this end, we started our efforts to achieve integration of existing development tools on XMs and PPSs in order to come up with a new tool that will be able to initially animate $OPERAS_{XC}$ specified models. The integration of the necessary features of these two tools into one will allow us to gain a deeper understanding of the modelling issues involved in engineering agent societies with $OPERAS_{XC}$ and help us investigate the practicability of our approach.

Acknowledgements

The authors would like to thank the reviewers for their valuable initial and additional comments as well as the Hellenic Artificial Intelligence Society for funding our participation to the ESAW 2007 workshop.

References

[1] Mamei, M., Menezes, R., Tolksdorf, R., Zambonelli, F.: Case studies for self-organization in computer science. Journal of Systems Arch. 52, 443–460 (2006)

[2] Dignum, V., Dignum, F.: Understanding organizational congruence: Formal model and simulation framework. In: Proceedings of the Agent-Directed Simulation Symposium (ADS 2007), Norfolk, USA (March 2007)

[3] Dignum, V., Dignum, F.: A logic for agent organization. In: Proceedings of the Workshop on Formal Approaches to Multi-Agent Systems Durham, September 3-7 (2007)

[4] Hoogendoorn, M., Schut, M.C., Treur, J.: Modeling decentralized organizational change in honeybee societies. In: Almeida e Costa, F., Rocha, L.M., Costa, E., Harvey, I., Coutinho, A. (eds.) ECAL 2007. LNCS (LNAI), vol. 4648, pp. 615–624. Springer, Heidelberg (2007)

[5] Charrier, R., Bourjot, C., Charpillet, F.: Deterministic nonlinear modeling of ant algorithm with logistic multiagent system. In: Proceedings of the 6th international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007). ACM, New York (2007)

[6] Matson, E., DeLoach, S.: Formal transition in agent organizations. In: Proceedings of the International Conference on Integration of Knowledge Intensive Multi-Agent Systems, pp. 235–240 (2005)

[7] DeLoach, S.A.: Engineering organization-based multiagent systems. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds.) SELMAS 2005. LNCS, vol. 3914, pp. 109–125. Springer, Heidelberg (2006)

[8] dInverno, M., Luck, M., Georgeff, M., Kinny, D., Wooldridge, M.: The dMARS architechure: A specification of the distributed multi-agent reasoning system. Autonomous Agents and Multi-Agent Systems 9, 5–53 (2004)

[9] Rabinovich, Z., Rosenschein, J.S.: Dynamics based control: Structure. In: Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains, at The 5th International Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan, pp. 148–161 (2006)

[10] Luck, M., d'Inverno, M.: Formal methods and agent-based systems. In: Rouff, C., Truszkowski, M.H.J.R.J., Gordon-Spears, D. (eds.) NASA Monographs in Systems and Software Engineering. Springer, Heidelberg (2006)

[11] Fisher, M., Wooldridge, M.: On the formal specification and verification of multi-agent systems. International Journal of Cooperating Information Systems 6, 37–65 (1997)

[12] Hilaire, V., Koukam, A., Gruer, P., Müller, J.P.: Formal specification and prototyping of multi-agent systems. In: Omicini, A., Tolksdorf, R., Zambonelli, F. (eds.) ESAW 2000. LNCS (LNAI), vol. 1972, pp. 114–127. Springer, Heidelberg (2000)

[13] Odell, J., Parunak, H.V.D., Bauer, B.: Extending UML for agents. In: Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence, pp. 3–17 (2000)

[14] Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multiagent systems. In: Giorgini, P., Müller, J.P., Odell, J.J. (eds.) AOSE 2003. LNCS, vol. 2935, pp. 214–230. Springer, Heidelberg (2004)

[15] Gutknecht, O., Ferber, J.: MadKit: a generic multi-agent platform. In: Proc. of the 4th International Conference on Autonomous Agents, pp. 78–79 (2000)

[16] Chopra, A.K., Mallya, A.U., Desai, N.V., Singh, M.P.: Modeling flexible business processes. In: AAMAS 2004 (2004)

[17] Krishna, S.N., Păun, G.: P systems with mobile membranes. Natural Computing: an international journal 4, 255–274 (2005)

[18] Cardelli, L., Gordon, A.D.: Mobile ambients. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378, pp. 140–155. Springer, Heidelberg (1998)

[19] Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes I. Information and Computation 100, 1–40 (1992)

[20] Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61, 108–143 (2000); Also circulated as a TUCS report since (1998)

[21] Banatre, J., Le Metayer, D.: The gamma model and its discipline of programming. Science of Computer Programming 15, 55–77 (1990)

[22] Berry, G., Boudol, G.: The chemical abstract machine. Journal of Theoretical Computer Science 96, 217–248 (1992)
[23] Rouf, C., Vanderbilt, A., Truszkowski, W., Rash, J., Hinchey, M.: Verification of NASA emergent systems. In: Proceedings of the 9th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2004), pp. 231–238 (2004)
[24] Stamatopoulou, I., Kefalas, P., Gheorghe, M.: Modelling the dynamic structure of biological state-based systems. BioSystems 87, 142–149 (2007)
[25] Kefalas, P., Stamatopoulou, I., Gheorghe, M.: A formal modelling framework for developing multi-agent systems with dynamic structure and behaviour. In: Pěchouček, M., Petta, P., Varga, L.Z. (eds.) CEEMAS 2005. LNCS (LNAI), vol. 3690, pp. 122–131. Springer, Heidelberg (2005)
[26] Stamatopoulou, I., Kefalas, P., Gheorghe, M.: Specification of reconfigurable MAS: A hybrid formal approach. In: Antoniou, G., Potamias, G., Spyropoulos, C., Plexousakis, D. (eds.) SETN 2006. LNCS (LNAI), vol. 3955, pp. 592–595. Springer, Heidelberg (2006)
[27] Stamatopoulou, I., Kefalas, P., Gheorghe, M.: OPERAS$_{CC}$: An instance of a formal framework for MAS modelling based on Population P Systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 551–566. Springer, Heidelberg (2007)
[28] Eilenberg, S.: Automata, Languages and Machines. Academic Press, London (1974)
[29] Kefalas, P., Eleftherakis, G., Kehris, E.: Communicating X-machines: A practical approach for formal and modular specification of large systems. Journal of Information and Software Technology 45, 269–280 (2003)
[30] Eleftherakis, G.: Formal Verification of X-machine Models: Towards Formal Development of Computer-based Systems. PhD thesis, Department of Computer Science, University of Sheffield (2003)
[31] Holcombe, M., Ipate, F.: Correct Systems: Building a Business Process Solution. Springer, London (1998)
[32] Bernandini, F., Gheorghe, M.: Population P Systems. Journal of Universal Computer Science 10, 509–539 (2004)
[33] Kefalas, P., Eleftherakis, G., Holcombe, M., Stamatopoulou, I.: Formal modelling of the dynamic behaviour of biology-inspired agent-based systems. In: Gheorghe, M. (ed.) Molecular Computational Models: Unconventional Approaches, pp. 243–276. Idea Publishing Inc. (2005)
[34] Rouff, C., Vanderbilt, A., Hinchey, M., Truszkowski, W., Rash, J.: Properties of a formal method for prediction of emergent behaviors in swarm-based systems. In: Procedings of the 2nd International Conference on Software Engineering and Formal Methods, pp. 24–33 (2004)
[35] Kefalas, P., Stamatopoulou, I., Gheorghe, M.: Principles of transforming Communicating X-machines to Population P Systems. In: Proceedings of the International Workshop on Automata for Cellular and Molecular Computing (ACMC 2007) (2007); Also to appear in the International Journal of Foundations of Computer Science