2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems Workshops

Run-time Verification of MSMAS Norms Using Event Calculus

Emad Eldeen Elakehal
Department of Computer Science
University of Bath, BATH, UK
Email: emad@itu.dk

Marco Montali
KRDB Research Centre
Free University of Bozen-Bolzano, Italy
Email: montali@inf.unibz.it

Julian Padget
Department of Computer Science
University of Bath, UK
Email: jap@cs.bath.ac.uk

***Abstract*—Modelling Self-managing Multi Agent Systems (MSMAS) is a software development methodology that facilitates designing and developing complex distributed systems based on the multiagent systems paradigm. MSMAS uses a declarative modelling style to capture system requirements by specifying four types of what we call *system* norms over: the system goals, the system roles, the business activities, and communications. MSMAS utilises system norms to capture system requirements in a formal language which can subsequently be monitored and verified at runtime. In this paper we present the main elements of MSMAS and introduce MSMAS defined norm types. We model the life cycle of MSMAS norms as non-atomic activities and formally express them as Event Calculus (EC) theories. Our axiomatisation of MSMAS system norms as first-order EC allows for reasoning with a metric time representation which we illustrate through a monitoring example of two execution traces to verify the system compliance with its intended design requirements and show how to detect any violation of norms.**

I. Introduction

Modelling Self-managing Multi Agent Systems (MSMAS) is a recently proposed methodology [1], [2] that covers the full development life cycle and aims at providing modelling steps and processes to analyze, design and implement Multi Agent Systems (MAS), with a particular focus on business-oriented applications. We propose MSMAS to bridge a gap between theory and practice and to address a number of issues with other available MAS development methodologies that, in our view, have impeded the commercial up-take of the MAS paradigm. MSMAS combines business process and agent concepts and defines notations for visual models with underlying formal presentations. The MSMAS approach provides tool support and improves access for business users, to model and develop agent-based smart business processes [3].

MSMAS incorporates the concept of norms and so MSMAS developed MASs can be called Normative Multi Agent Systems (nMAS). These are systems with an explicit representation of norms, meaning they are formalised and codified in some form by an authority, and norms can be violated meaning the MAS must have the possibility of deviating in its actual course of action from the ideal one.

MAS in general and nMAS in particular are open systems that offer an environment where the system components enjoy a high degree of flexibility and freedom in deciding on their own course of actions. As a result, agents cannot be assumed to behave at all times in compliance with the system goals. The MSMAS approach enables the system developer to capture and encode system requirements as norms. This allows not only for model checking during design time, but also for monitoring and verifying the system execution during run-time. This paper outlines the MSMAS modelling approach and the mapping of MSMAS norms to Event Calculus EC to enable reasoning about system execution.

The paper is structured as follows, Section II contains a brief overview of the MSMAS phases, a description of how norms are expressed in MSMAS models, and the modelling of MSMAS event and norm transitions as non-atomic activities. Section III starts with a summary of the Event Calculus, followed by the formal representation of MSMAS norms as EC domain-specific theories. Section IV illustrates by an example how to monitor and verify the execution traces of deployed MSMAS systems at run-time, and we conclude in Section V with discussion and directions for future work.

II. MSMAS Methodology Overview

MSMAS is a comprehensive methodology consisting of three phases that cover the full development life cycle of MASs. A summary of MSMAS phases is presented below (see [1] and [4] for full details):

1) The first phase gathers **System Requirements**, where the system designer drafts the main use case scenarios and specifies, at a high-level, the system goals.
2) The second phase is **Detailed Analysis and Design**, where the requirements are transformed into a complete system model. It has two stages: (i) the **Initial System Design** stage defines precise relations between the system goals and specifies the system organisational structure, and (ii) the **Detailed Design Stage**, comprising the design of *Business Activities* and full specification of the *System Participants* and the *Communication Protocols*.
3) The third phase deals with **Verification and Implementation**, in which the specification is exported in one of the two available formats (CLIMB/*S*CIFF [5] or Resource Description Framework (RDF) [1])

A. Norms in MSMAS

MSMAS uses norms as a core concept to capture the system requirements in a formal language which can subsequently be verified statically and monitored at runtime. MSMAS norms place constraints on system participants' behaviour. We

[1] http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

978-1-4799-6378-2/14 $31.00 © 2014 IEEE
DOI 10.1109/SASOW.2014.31

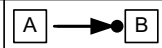
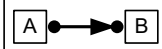
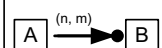
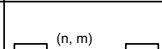
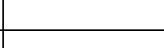
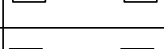
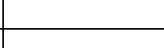
ConDec++ Notation	ConDec++ Visual notation	MSMAS Goal/Goal Relation
Precedence Relationship: If ***B*** Is performed ***A*** should Have been performed before it		Sequential Goals: ***Goal B*** has to be achieved only after achieving ***Goal A***
Succession Relationship: every execution of ***A*** should be followed by the execution of ***B*** and each ***B*** should be preceded by ***A***		Joint Goals: ***Goal B*** must be achieved after achieving ***Goal A,*** and ***Goal A*** must be achieved before achieving ***Goal B***
Precedence Relationship with Time Constraint		Time Constrained Sequential Goals: The system participant has to achieve ***Goal B*** after minimum delay of n and maximum delay of m after achieving ***Goal A***
Succession Relationship with Time Constraint		Time Constrained Joint Goals: ***Goal B*** must be achieved after minimum delay of *n* and maximum delay of *m* after achieving ***Goal A,*** and ***Goal A*** must be achieved before achieving ***Goal B***
Coexistence Relationship: If either ***A*** or ***B*** is performed, the other one has to be executed as well.		Coupled Goals: Both ***Goal A*** and ***Goal B*** must be achieved
Not Coexistence Relationship: If one of ***A*** or ***B*** is performed, the other one can not be executed.		Disjoint Goals: If ***Goal A*** has been achieved then ***Goal B*** can not be achieved, and vice versa.
No constraint		Amicable Goals: Any or both of ***Goal A*** and ***Goal B*** can be achieved without any restriction

Fig. 1. ConDec^{++} Notation and its Mapping to MSMAS Message/Message and Goal/Goal relation concepts

express MSMAS norms visually using ConDec^{++} which is an extension of the DECLARE [2] language, developed by van der Aalst and Pesic [6], [7].

The modelling of MAS as normative/constraint-based systems, allows the system designer to place constraints on business activities, system participant institutional roles, system goals, and communication messages. The specifications are defined through explicit visual relations expressed in the ConDec^{++} language, whose semantics we extend to define four types of norms/sets of constraints based on the context of the visual model. For example, the ConDec^{++} *Precedence Relationship* between two business activities, states that "If activity B is performed, activity A should have been performed before it", and we use the same graphical notation to present the sequential goal/goal relation in the composite business process model, which states that goal A has to be satisfied before starting to achieve goal B. Figure 1 lists the ConDec^{++} notations and their mapping to MSMAS goal/goal-relation concepts. Other norm types are role/role relations, message/message relations and activity/activity relations, which are not included due to the limited paper length.

We use System Norms (SN) to model behavioural patterns of system participants and their inter-relations at the social level, including messages to exchange, roles to play, goals to achieve and activities to execute. A system norm is written:

$$SNGroup(A, B, SNType).$$
$$SNGroup(A, B, (SNType, (t_{min}, t_{max}))).$$

where A and B are the constrained system entities, such as a pair of *System Goals*, *Business Activities*, *Communication Messages*, or *Institutional Roles*. The $SNGroup$ is the name of the SN group. We define four system norm groups, one for each system entity type: *goal_goal_relation*, *activity_activity_relation*, *message_message_relation*, or *role_role_relation*. The $SNType$ is the type of relation that holds between a pair of entities such as: *sequential_goals*, *joint_goals*, (tc denotes time constrained) *tc_sequential_goals*, *tc_joint_goals*, *coupled_goals*, or *disjoint_goals*, e.g:

$$goal\_goal\_relation(BSG\_Collect\_Data, BSG\_Enrol\_User, (tc\_sequential\_goals, (10, 50))).$$

expresses that a time-constrained sequential goals relation holds between $BSG\_Collect\_Data$ and BSG_Enrol_User goals with minimum delay of 10 time units and maximum delay of 50 time units. This norm represents a system requirement which states that the system has to enrol the user after collecting the data within the time window indicated.

B. MSMAS Activity Life cycle and State Transition Triggers

Achieving a system goal or playing institutional roles in MSMAS is considered a non-atomic durative activity where the activity execution spans a time period according to a life cycle and is motivated by multiple event occurrences. We segregate MSMAS sets of events which affect the system norms into two groups: (i) system goals (SG) and business activities (BA) events, and (ii) institutional roles (IR) and communication messages (CM) events.

We then model the activity life cycle that governs the first group (system goals and business activities-related events) using only three states: $active$, $inactive$, and $complete$ (Figure 2(a)). While we define another simpler life cycle of only two states: $active$, and $inactive$ (Figure 2(b)) that governs institutional roles and communication messages-related actions. The second life cycle has fewer states because some of MSMAS activities are binary: consider Institutional Roles, for example, where a system participant either plays the role or not. The system execution events may have triggering events that implicitly manipulate each non-atomic activity instance causing it to change state.

Triggering events are those events that cause the activity instance to move from one state to another. For example the system goals activity triggering events are expressed as:

$$Event(GoalEventType(SGID, SPID), T).$$

where $GoalEventType$ can be: $achieve$, $satisfy$, or $drop$ goal. $SGID$ is a system goal ID, $SPID$ is the ID of the system participant, and T is the time stamp of the event.

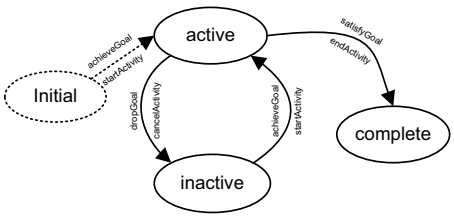
The system designers, through their design, implicitly define which events trigger and motivate the state transitions of each activity. The full list of triggering events is:

System Goals:	$(achieve - satisfy - drop)$
Business Activities:	$(start - complete - cancel)$
Institutional Roles:	$(playRole - dropRole)$
Communication Messages:	$(sendMessage - cancelMessage)$

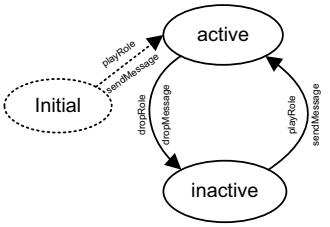
III. MSMAS Norm Modelling in the Event Calculus

One of the most critical tasks of developing adaptive self-managed systems, is to verify that the system will not fail to meet the requirements once developed and deployed.

[2]http://www.win.tue.nl/declare/, retrieved 09 July 2014



(a) SG and BA Life Cycle



(b) IR and CM Life Cycle

Fig. 2. MSMAS Events and Goals, Activities, Roles, Messages Generic Life Cycle Modelled as Non-atomic Activities

$happens(Ev, T)$	Event Ev happens at time T
$hold\_at(F, T)$	Fluent F holds at time T
$hold\_for(F, [T_1, T_2])$	Fluent F holds during the interval $[T_1, T_2]$
$initially(F)$	Fluent F holds in the initial state of the system
$initiates(Ev, F, T)$	Event Ev initiates Fluent F at time T
$terminates(Ev, F, T)$	Event Ev terminates Fluent F at time T

TABLE I. THE EVENT CALCULUS ONTOLOGY

Although validation of the system models is a good way to check if the model complies with the requirements, it does not guarantee that the system will comply at run-time, hence run-time verification is required. Validation and verification of system models and its execution traces require the formal representation of the system and the use of mathematical and/or logical techniques. We chose EC for our formal representation of norms, because it allows for explicit representation of time, making it possible to express both qualitative and quantitative time constraints and being based on first-order logic it provides great expressiveness. In this section, we start with a brief summary of the $EventCalculus$, followed by an illustration of the mapping of MSMAS norms into EC domain-independent and domain-specific theories.

A. The Event Calculus

Kowalski and Sergot [8] proposed Event Calculus (EC) as a general framework to reason about time dependent properties called $fluents$, and $events$ that affect these fluents over time. The general theory axiomatising EC that defines the meaning of the predicates supported by the calculus is called the EC **Ontology** and it contains the set of predicates shown in Table I. An EC theory is a way to formalise how domain-specific events affect the domain-specific fluents, which can then be constituted as a logic program whose clauses define the system initial state and relate events occurrences with fluents initiation and termination [9].

We express MSMAS norms by two different theories: (i) a domain-independent theory, that formalises the life cycle of time-aware system norms (Figures 2(a), and 2(b)), and gives semantics to the state transitions by relating the initiation/termination of norm-related fluents, and (ii) a theory that describes a specific domain, and defines the relation between the domain-specific events and fluents/norms.

B. Domain-independent Theory of MSMAS Activity Life Cycle

When an event occurs, it may affect MSMAS activities and cause the system to move from its current state to a new state according to the activity life cycles shown in Figures 2(a) and 2(b). We follow the formalisation proposed by Montali et al. [9] and use tools to support the monitoring of event based systems[3]. We use the fluent $ai\_state(i(id, a), s)$ to denote that an activity instance identified by i of system component a is currently in state s.

Axiom 1 (Effective Start). *An activity instance is effectively $started$ (s) by a start event occurrence as long as it is not already started. The opposite state $\neg started$ is defined in Axiom 3*:

$$happens(start(ID, A), T) \leftarrow happens(ev(ID, s, A), T) \\ \wedge \neg initiates(ev(ID, s, A), ai\_state(i(ID, A), active), T).$$

The effective start triggers a creation of the corresponding activity instance, transferring the identifier and placing the instance in the active state:

$$initiates(start(ID, A), ai\_state(i(ID, A), active), T).$$

Axiom 2 (Effective Completion). *An activity instance with name A and identifier ID is effectively completed (c) at time T if a completion event matching A and ID occurs at some time T, such that the activity instance is $active$ at time T*:

$$happens(complete(ID, A), T) \leftarrow happens(execute(ID, c, A), T) \\ \wedge holds\_at(ai\_state(i(ID, A), active), T).$$

Effective completion triggers transition to the completed state:

$$terminates(complete(ID, A), ai\_state(i(ID, A), active), T). \\ initiates(complete(ID, A), ai\_state(i(ID, A), complete), T).$$

Axiom 3 (Effective Cancellation). *The effective cancellation n of an activity instance mirrors the axioms used for effective start as its semantics is $\neg started$*:

$$happens(cancel(ID, A), T) \leftarrow happens(execute(ID, n, A), T) \\ \wedge holds\_at(ai\_state(i(ID, A), active), T) \\ \wedge \neg initiates(ev(ID, n, A), ai\_state(i(ID, A), inactive), T).$$

Effective Cancellation triggers transition to the inactive state:

$$terminates(cancel(ID, A), ai\_state(i(ID, A), active), T). \\ initiates(cancel(ID, A), ai\_state(i(ID, A), inactive), T).$$

C. MSMAS Norm Instances and their States

The triggering events defined earlier not only have an impact on the states of their associated activities but their effect extends to each system norm associated with such activities. For example assuming a $joint\_goals$ relation holds between $SG1$ and $SG2$ system goals, this norm is triggered every time a system participant successfully satisfies/achieves $SG1$ goal. That is, the goal activity instance reaches the state $complete$, expecting the completion of the target goal $SG2$ at some point in the future. To capture this behaviour, every execution that satisfies the source goal creates a fresh instantiation of the norm and this particular norm instance is placed in a $pending$ state, waiting for the occurrence of the target goal. In this work,

[3]A number of tools including jREC reasoner available via https://www.inf.unibz.it/~montali/tools.html, retrieved 10 July 2014

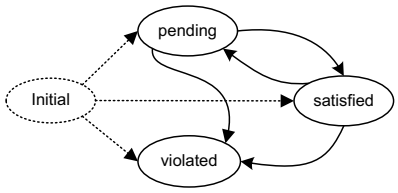


Fig. 3. MSMAS System Norms Generic Life Cycle Modelled as Non-atomic Activities

$initially(sn\_state(I, S)) \leftarrow init\_state(I, S).$
$initiates(E, sn\_state(I, S), T) \leftarrow creation(E, T, I, S).$
$cur\_state(I, S, T) \leftarrow holds\_at(sn\_state(I, S), T).$
$terminates(E, sn\_state(I, S_1), T) \leftarrow trans(E, T, I, S_1, S_2).$
$initiates(E, sn\_state(I, S_2), T) \leftarrow trans(E, T, I, S_1, S_2)$
$\wedge holds\_at(sn\_state(I, S_1), T).$

Fig. 4. State manipulation predicates [9]

we present an example of MSMAS norms to demonstrate how they can be modelled. We associate a unique identifier with each modelled norm, the term $i(id,\ a,\ t)$ denotes the instance of norm id has been created by the execution of a at time t. Each norm instance can be in only one state at any point of time and it obeys the generic life cycle of MSMAS norms as shown in Figure 3, where $pending$ is a transient state meaning that the SN is waiting for the occurrence of some event, $satisfied$ is either a transient or a permanent state indicating that the execution trace is currently compliant with the SN, and $violated$ is a permanent state[4] indicating that the instance has been violated.

We employ a multi-valued fluent $sn\_state(I,\ S)$ to present the state of each system norm instance, where I is the SN instance and S is the current state of I which can be one of the possible three values ($pending$ - $satisfied$ - $violated$). For example, $sn\_state(i(id,\ a,\ t),\ satisfied)$ represents the fact that SN instance $i(id,\ a,\ t)$ is currently satisfied.

D. Formalising MSMAS Norms (Domain-Specific Theory)

First we present our axiomatisation of MSMAS SNs as an EC theory. We need both domain-specific theories with the domain-independent theories defined in section III-B to reason about the execution traces and to verify if the system complies/violates its specified requirement at run-time. Our formalism depends on the creation of, and the state transitions of, SNs instances, as a result of event occurrences. To support the state manipulation, we use the five predicates defined by Montali et al. [9], where the first two rules deal with the creation of a SN instance and setting its state. The third predicate is used to check the current state of a SN instance and the last two rules capture the state transitions (Figure 4).

Now let us consider SN I, the Metric Time Constraint Sequential Roles System Norm, expressed formally in terms of EC theories, where Figure 5 shows its life cycle. In this type of directional system norm between two components (e.g. institutional roles A and B), where A is the source institutional role (IR), the system participant has to play the target role after it plays the source role with minimum delay (n) and maximum

[4]The recovery from the state $violated$ might be desired by particular applications, but it is not considered in this work.

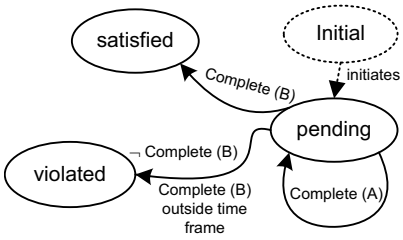


Fig. 5. SN I : MSMAS Norm Life Cycle Modelled as Non-atomic Activity

delay (m) time units. For example:

$role\_role\_relation(IR\_TeamLead, IR\_Manager,$
$(tc\_sequential\_roles, (1500, -))).$

states that a Metric Time Constrained Sequential Roles relation holds between $IR\_TeamLead$ and $IR\_Manager$ institutional roles, expressing the requirement that a system participant can be a manager only after being a team lead for a time period of 1500 time units at least. To express this kind of system norm in EC we define the following axioms.

Axiom 4 (TC Sequential Institutional Roles Creation). *Each sequential institutional roles system norm is associated with a unique instance, created and put in the pending state waiting for the completion of playing role A when the case is started*:

$$init\_state(i(id, start, 0), pend).$$

Axiom 5 (TC Sequential Institutional Roles Pending). *An initially pending TC Sequential Institutional Roles SN instance remains in the state $pending$ when its source IR is played (completed). This self transition is done only to set the start time. This reference point is combined with the norm's delay n and deadline m to determine the time window, inside which the target role is expected to be played and against which we can determine the violation or satisfaction of the norm condition*:

$trans(complete(-, A), T, i(id, A, T_i), pend, pend) \leftarrow$
$cur\_state(i(id, A, T_i), pend, T) \wedge T \geq T_i.$

Axiom 6 (TC Sequential Institutional Roles Fulfilment). *A pending TC Sequential Institutional Roles SN instance is satisfied when its target institutional role is played within the time frame specified*:

$trans(complete(-, B), T, i(id, A, T_i), pen, sat) \leftarrow$
$cur\_state(i(id, A, T_i), pend, T) \wedge T \geq T_i + n \wedge T \geq 0.$

The absence of a deadline means it is effectively ∞. We check also that $T > 0$, as $T = 0$ indicates that pending is the initial state, set during the creation of this instance.

Axiom 7 (TC Sequential Institutional Roles Violation). *A pending TC Sequential Institutional Roles SN instance is violated when: (i) the target role is played before the source role, or (ii) the target role is played outside the time frame specified by the minimum and maximum delay values (n, m). In our example the maximum time delay is not specified so we limit this axiom to the first case*:

$trans(complete(-, B), T, i(id, A, T_i), pend, viol) \leftarrow$
$cur\_state(i(id, A, T_i), pend, T) \wedge T < T_i + n.$

Axiom 8 (Violation due to Deadline Expiration). *In the case of time constrained SN, the SN is violated if it is still pending after the maximum delay time passes without the completion of the target specified activity such as playing a role or sending a message ... etc. This axiom handles this case and is valid*

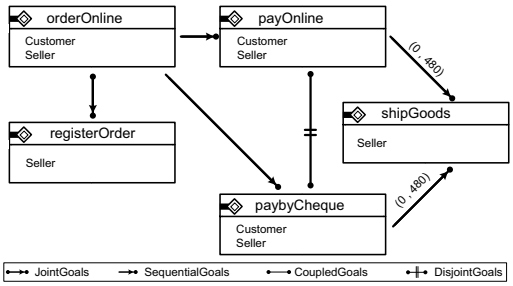


Fig. 6. Example MSMAS Composite Business Process Model with Goal/Goal relations

only when the maximum time delay is specified:

$$trans(-, T, i(id, A, T_i), pend, viol) \leftarrow$$
$$cur\_state(i(id, A, T_i), pend, T) \wedge T > T_i + m.$$

IV. Monitoring Example

Let us consider the composite business process model shown in Figure 6. There are five Basic Business Processes, each corresponding to a Basic System Goal. This business process is a segment of an ordering system that supports the selling of goods online and offers the customer the option to pay either online or by sending a cheque. The model is designed to capture the following business requirements:

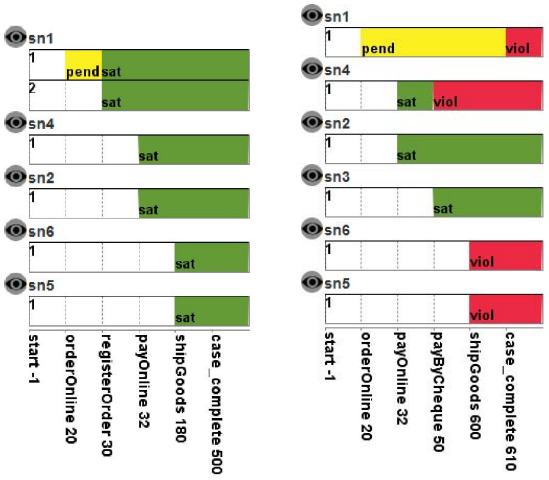
R1: After the customer communicates with the seller to place an order, the seller has to register the customer's order.
R2: The customer can pay the total selling price of his/her order either by credit card online, or by posting a cheque.
R3: The customer pays online *only* after he/she has ordered.
R4: If the customer pays online then he/she cannot pay by cheque and vice versa to prevent duplicate payment.
R5: After completion of payment, the seller must ship the goods to the customer within at most 480 time units.

These business requirements are captured and modelled by means of goal/goal system norms, expressed visually as six relations between the business process pairs as follows:

SN1: **JointGoals** relation between *orderOnline* and *registerOrder* to capture R1.
SN2: **sequentialGoals1** relation between *orderOnline* and *payOnline* to capture R2 and R3.
SN3: **sequentialGoals2** relation between *orderOnline* and *paybyCheque* to capture R2 and R3.
SN4: **DisjointGoals** relation between *payOnline* and *paybyCheque* to capture R4.
SN5: **time constrained sequentialGoals1** relation between *payOnline* and *shipGoods* to capture R5.
SN6: **time constrained sequentialGoals2** relation between *paybyCheque* and *shipGoods* to capture R5.

Now let us consider the following execution traces[5], where the first trace satisfies the requirements:

[5]Due to the size of this paper we show only the completion events e.g. satisfyGoal the full trace should contain achieveGoal events as well.



(a) System Norms Evolution According to Execution Trace 1

(b) System Norms Evolution According to Execution Trace 2

Fig. 7. Comparison of System Norm Evolution in Trace 1 and Trace 2

Trace 1:

$$satisfy\_goal(orderOnline, sellerAgent), 20$$
$$satisfy\_goal(orderOnline, customerAgent), 21$$
$$satisfy\_goal(registerOrder, sellerAgent), 30$$
$$satisfy\_goal(payOnline, sellerAgent), 31$$
$$satisfy\_goal(payOnline, customerAgent), 32$$
$$satisfy\_goal(shipGoods, sellerAgent), 180$$

Here, the customer completes orderOnline goal at time 21, then payOnline at time 32 which satisfies R2 and R3. The seller completes orderOnline at time 20, then registerOrder at time 30 which satisfies R1. The seller completes payOnline at time 31 which satisfies R2 and R3. Finally, the seller completes shipGoods at time 180 which satisfies R5. Neither the seller nor the customer start and complete paybyCheque goal within this trace, which satisfies R4. Figure 7(a) shows[6] the list of system norms and their state evolution according to execution trace 1. But the second trace violates the requirements:

Trace 2:

$$satisfy\_goal(orderOnline, sellerAgent), 20$$
$$satisfy\_goal(orderOnline, customerAgent), 21$$
$$satisfy\_goal(payOnline, sellerAgent), 31$$
$$satisfy\_goal(payOnline, customerAgent), 32$$
$$satisfy\_goal(paybyCheque, customerAgent), 50$$
$$satisfy\_goal(paybyCheque, sellerAgent), 50$$
$$satisfy\_goal(shipGoods, sellerAgent), 600$$

Here, the customer completes orderOnline goal at time 21 and payOnline at time 32, which satisfies R2 and R3. Then the customer completes paybyCheque goal at time 50 which violates SN4/R4. The seller completes orderOnline goal at time 20, then payOnline at time 31 which satisfies R2 and R3. The seller completes paybyCheque at time 50 which violates SN4/R4. The seller completes shipGoods at time 600, which violates R5 because it occurs after the maximum time delay allowed by the time constrained sequential SN. Finally, the seller never completes registerOrder within this trace, so it is considered violated. Figure 7(b) shows the list of system norms and their state evolution according to execution trace 2.

[6]Generated using MOBUCON tool (https://www.inf.unibz.it/~montali/tools.html, retrieved 31 Jan 2014), using a reduced version of events (limited completion events e.g. satisfyGoal only)

V. Discussion and Future work

We present an extensive survey of MAS development methodologies in [1] and for the sake of space do not reproduce those arguments to make the case for MSMAS here. We also observe that while the work here is presented in the context of MSMAS, the focus of this paper is less on the framework of MSMAS, but the capture of requirements in formal language, their verification and their run-time monitoring. Hence, these are the areas from which we draw related work.

The main goal of modelling MASs with norms is to use normative models to govern agent behaviour, and to model social structure and its evolution [10]. In this context, there is a wealth of research on formalising normative systems, including [11] and [12]. Monitoring agent compliance with institutional rules/system norms has been studied in [13] and more recently in [14], which presents a formalisation of the life cycle of regulative and constitutive norms and a supporting reasoner. MSMAS uses norms in a similar way for the static checking of specifications – although the formalisation and tools are different - but in addition, it uses those same normative specifications to verify the maintenance of system requirements during execution, where the models are derived from the requirements, and system goals are linked to business activities associated with institutional roles.

The importance of verifying requirements is widely recognised given the increased complexity of MAS and increased dynamic nature of business requirements. Some approaches focus on traceability from requirements to source code and/or between design and source code [15], while others focus on traceability between requirements and architecture [16], allowing for generation and validation of traces using requirements relations. In contrast, MSMAS utilises logic-based languages to encode formally system norms that represent system requirements into models that become an integral part of the deployment. Hence, traceability of these requirements from design time to verify the correctness of the designed models, as well as monitoring the execution traces and verifying that the trace meets the requirements, are achievable tasks.

Chesani *et al* [17] propose monitoring and runtime compliance checking using an abductive logic programming-based proof procedure, then in [18] describe the EC-based axiomatisation of ConDec. We have adopted their approach due to the availability of scalable support tools ready for integration.

Against this background – and a larger body of literature than can be cited or discussed here – we suggest that MSMAS use of system norms in the context of building business oriented MAS and its support for design- and run-time formal verification provides a new perspective on formal software engineering of MAS and contributes towards the evolution of model-driven development in MAS. Primary tasks for future work for MSMAS are to incorporate the modelling of multiple organisations/institutions and their interactions, and to consider how to incorporate attributes of the context into the modelling to allow for the capture of more fine-grained constraints.

References

[1] E. E. Elakehal and J. Padget, "MSMAS: Modelling self-managing multi agent systems," *SCPE: Scalable Computing: Practice and Experience*, vol. 13, no. 2, pp. 121–137, 2012.

[2] ——, "A practical method for developing multi agent systems: APMDMAS," in *Intelligent Distributed Computing V*, ser. Studies in Computational Intelligence, F. B. et al, Ed. Springer, 2012, vol. 382, pp. 11–20. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24013-3_3

[3] J. Sinur, J. J. Odell, and P. Fingar, *Business process management: the Next Wave*. Meghan-Kiffer Press, 2013.

[4] E. E. Elakehal, M. Montali, and J. Padget, "Verifying msmas model using $\mathcal{S}$ciff," in *Multiagent System Technologies*, ser. LNCS, M. Klusch, M. Thimm, and M. Paprzycki, Eds. Springer Berlin Heidelberg, 2013, vol. 8076, pp. 44–58. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40776-5_7

[5] M. Montali, M. Pesic, W. M. P. v. d. Aalst, F. Chesani, P. Mello, and S. Storari, "Declarative specification and verification of service choreographiess," *ACM Trans. Web*, vol. 4, no. 1, pp. 3:1–3:62, Jan. 2010. [Online]. Available: http://doi.acm.org/10.1145/1658373.1658376

[6] W. van der Aalst and M. Pesic, "DecSerFlow: Towards a truly declarative service flow language," in *The Role of Business Processes in Service Oriented Architectures*, ser. Dagstuhl Seminar Proceedings, F. L. et al, Ed., no. 06291, Dagstuhl, Germany, 2006. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2006/829

[7] M. Pesic and W. Aalst, "A declarative approach for flexible business processes management," in *Business Process Management Workshops*, ser. LNCS, J. Eder and S. Dustdar, Eds. Springer Berlin Heidelberg, 2006, vol. 4103, pp. 169–180. [Online]. Available: http://dx.doi.org/10.1007/11837862_18

[8] R. A. Kowalski and M. J. Sergot, "A logic-based calculus of events," *New Generation Comput.*, vol. 4, no. 1, pp. 67–95, 1986.

[9] M. Montali, F. M. Maggi, F. Chesani, P. Mello, and W. M. van der Aalst, "Monitoring business constraints with the event calculus," *ACM Transactions on Intelligent Systems and Technology*, 2011.

[10] T. Balke, C. da Costa Pereira, F. Dignum, E. Lorini, A. Rotolo, W. Vasconcelos, and S. Villata, "Norms in MAS: Definitions and related concepts." in *Normative Multi-Agent Systems*, 2013, pp. 1–31.

[11] A. García-Camino, P. Noriega, and J. A. Rodríguez-Aguilar, "Implementing norms in electronic institutions," in *Proc. 4th Intl Joint Conf. on Autonomous Agents and Multiagent Systems*. ACM, 2005, pp. 667–673.

[12] G. Boella and L. der Torre, "Constitutive norms in the design of normative multiagent systems," in *Computational Logic in Multi-Agent Systems*, ser. LNCS, F. Toni and P. Torroni, Eds. Springer Berlin Heidelberg, 2006, vol. 3900, pp. 303–319. [Online]. Available: http://dx.doi.org/10.1007/11750734_17

[13] H. L. Cardoso and E. Oliveira, "Institutional reality and norms: Specifying and monitoring agent organizations," *International Journal of Cooperative Information Systems*, vol. 16, no. 01, pp. 67–95, 2007. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0218843007001573

[14] S. Alvarez-Napagao, H. Aldewereld, J. Vázquez-Salceda, and F. Dignum, "Normative monitoring: Semantics and implementation," in *Coordination, Organizations, Institutions, and Norms in Agent Systems VI*, ser. LNCS, M. Vos, N. Fornara, J. Pitt, and G. Vouros, Eds. Springer Berlin Heidelberg, 2011, vol. 6541, pp. 321–336. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-21268-0_18

[15] M. Grechanik, K. S. McKinley, and D. E. Perry, "Recovering and using use-case-diagram-to-source-code traceability links," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2007, pp. 95–104.

[16] A. Goknil, I. Kurtev, and K. Van Den Berg, "Generation and validation of traces between requirements and architecture based on formal trace semantics," *Journal of Systems & Software*, vol. 88, pp. 112–137, 2014.

[17] F. Chesani, P. Mello, M. Montali, and P. Torroni, "A logic-based, reactive calculus of events," *Fundamenta Informaticae*, vol. 105, no. 1, pp. 135–161, 2010.

[18] ——, "Web services and formal methods," R. Bruni and K. Wolf, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Verification of Choreographies During Execution Using the Reactive Event Calculus, pp. 55–72. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01364-5_4