

Verifying MSMAS Model Using SCIFF

Emad Eldeen Elakehal¹, Marco Montali², and Julian Padget¹

¹ Department of Computer Science
University of Bath, BATH BA2 7AY, UK
emad@itu.dk, jap@cs.bath.ac.uk

² KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy
montali@inf.unibz.it

Abstract. MSMAS is a software development methodology that facilitates the design and development of complex distributed systems based on the multiagent systems paradigm. MSMAS explicitly supports the institutional organisational structure and follows a declarative modelling style to specify behavioural restrictions on the members of the institution, their roles, the business processes regulating their behavior and the communication protocols regulating their mutual interactions. All these aspects are visually represented, by adapting the DECLARE graphical language, proposed for the declarative specification of constraint-based business processes. In this paper we discuss the main elements of MSMAS, and show how they can be equipped with a formal, expectation-based semantics, tailored to the SCIFF Abductive Logic Programming-based framework. In particular, we show how the MSMAS constructs can be formalized in SCIFF, and then exploit this correspondence to specify and verify formal properties over MSMAS models, by leveraging on the SCIFF reasoning capabilities.

1 Background

With a growing interest in modelling the social structure of modern distributed systems and increased research activities around using role norms and institutions as an organisational type to capture these social aspects, it seems that modelling efforts are disconnected from implementation at the application level. Some metamodels lack supporting design tools or if not they lack design verification and run-time validation capabilities. The MSMAS [6,5] methodology aims to establish a link between modelling and implementation by combining business oriented metamodeling with institution and role modelling and supports a formal proof mechanism for design and runtime validation. MSMAS allows multi agent system (MAS) designers to model self-managing MASs using visual graphic models. Here, we take self-managing to mean the ability of the system to recognize execution errors or undesired behaviour and the ability to respond by replanning in order to recover from the failure or to stop the undesired activities. MSMAS has three phases: the first phase is to capture the system requirements through the *Use Cases Models* and to create the *System Goals Model*. The second phase starts with the high level design of the required *Business Processes* to achieve the system goals and the specification of the system organisational structure through the *Institutions Models*. Then, a detailed design of the business activities, a full specification

of the *System Participants* and the specification of *Communication Protocols* that defines how system participants can coordinate their activities, interact with one another and exchange information. The third phase concerns implementation, where the user can export the system specification in either of the two available formats. The first is the SCIFF formal framework [1], which supports the designer in the assessment of the produced model, checking its correctness and verifying whether it meets desired properties, also taking into account possible execution traces produced by the system. The second format is RDF that offers a basis for transformation to any other execution languages such as JADE, JASON, etc. Or can be mapped to another RDFs such as the frame work proposed by Alberola et al[7]. Verifying SCIFF model is good indicator of the correctness of the RDF model, because both models are reflecting the same MSMAS metamodel.

In MSMAS, the system designer can set constraints on the business processes and/or their activities, as well as on the system participant roles. Any activity without a constraint can be executed an arbitrary number of times in an arbitrary order as long as its preconditions are satisfied, while the constraints on system participants' roles are used to specify the accepted behavioural patterns. To impose dynamic constraints on the activity execution, MSMAS uses the graphical notation of DECLARE (see <http://www.win.tue.nl/declare/>, retrieved 20130627), deriving from the DecSerFlow/ConDec languages developed by van der Aalst and Pesic [14,13,11]. DECLARE is a declarative language for modelling and enacting constraint-based business processes. We chose a declarative approach for MSMAS because it is well-suited to the dynamic nature of MASSs, and because DECLARE offers a simple graphical notation with a powerful and flexible formal representation.

DECLARE takes an open approach where the relationship constraints that are set between two or more activities can be either positive and negative. Positive relationships are normally used to state that a certain activity is expected to occur when a certain system state is achieved, while negative relationships state forbid the execution of activities when a given state of affairs holds. DECLARE offers a number of loosely-coupled template relations, which go beyond the standard sequential relationships of classical process specification languages. An example is the *responded presence* constraint, which states that if the source activity A is executed, then the target activity B must be executed as well, either before or after the execution of activity A.

Many formal models for agent-based systems have been proposed and they present useful approaches for building such systems. We believe MSMAS makes a valuable contribution to this line of research because of its ability to model scenarios, in which the system components and human participants interact governed by social norms. We consider this is an important aspect, as modelling only individual agent aspects cannot cover all the issues that affect how they interact and coordinate their behaviour to allow the system to achieve its goals; hence considering the social aspects of these individual agents becomes necessary. Incorporating social and organizational structure complicates the MAS model, however by following the MSMAS methodology and breaking down the system into smaller organisations and encoding the different behaviour patterns into roles, we argue that complication is contained. Furthermore, allowing the system properties to be assessed during design time with the support of a formal

framework helps in modelling societally-structured systems and the use of a declarative style enables monitoring and runtime verification [4,1,10].

2 Formal Model of MSMAS

Formal modelling methods of software comprise two activities: *formal specification* and *verification*. Formal specification permits the deployment of an accurate specification of the system behaviour that allows for a precise modelling of the system, while verification aims at proving that the model of the system complies with the intended requirements, and meets the desired properties.

Our evaluation of the literature and existing and past approaches, leads us to the conclusion that MSMAS can be based on just four concepts that are sufficient for a MAS description to be able to provide answers to the following four questions:

1. What is the purpose of building the system and its individual components? This question is answered by defining **System Goals** (SG) as the first core concept.
2. How can the system achieve its goals? The answer lies within the second core concept which is the system **Business Processes** (BP) and their activities.
3. Who or what is responsible for the execution of each business process activity? This is answered by the third core concept of **System Participants** (SP).
4. Through which organisational structures do the system participants interact and what roles can they play? This is answered by defining **System Institutions** (SI).

So, in our approach, a MAS is a 4-tuple: $MAS_{msmas} = \langle SG, BP, SP, SI \rangle$ and hence, in order to connect these four concepts, we address the modelling of: (i) System Participant Institutional Roles, (ii) System Participant Communication Protocols, and (iii) Business Processes Relationships. The system goals are the main drivers of the business processes and all activities in MSMAS are goal-directed so the formalisation above covers the complete MSMAS system view. In the rest of this section we go into more detail about each of the above concepts.

Institutional Roles. MSMAS requires the explicit statement of the society's organisational structure, where the system is organised in a number of institutions each of which has an associated finite set of roles, and system participants might play one or more of these roles in one or many of the system institutions. Specifying an organisation can be done through specifying the inter-agent relationships that exist within this organisation [15]. In MSMAS, these inter-relationships specifications are centered around the abstract roles the system participants can play and how these roles relate to each other. Role specification allows for defining behaviour patterns in an abstract way, independent from each individual system participant. In this sense, roles are considered as system participant types, so when a system participant takes part in an institution and plays one of the institution roles, it should conform to that pattern of behaviour. All system agents that adopt the same role are normally granted the same rights and duties, and are expected to obey to the same restrictions applied to that role. Declarative specification allows the identification of an arbitrary range of relations, but in MSMAS we restrict ourselves to the following role types, as seen in the role/role relations in Figure 1.

1. **Sequential Roles (SR):** these are the pairs of roles where the the system participant is required to play the first role before being allowed to play the second one. An example from Figure 1 is when an agent has to be *Catalogue Manager* before being *Stock Manager*.
2. **Joint Roles (JR):** these are pairs of roles where the system participant is required to play both, but one after another in a specified order, or neither. The first role is considered a precondition to place the second role. An example is the requirement in a marketplace that a type of seller should fulfill their customers' orders themselves, meaning they have to play the role of Shipper after playing being Seller.
3. **Coupled Roles (CR):** these are pairs of roles that are coupled together where the system participant is required to play both or none, but in either order: once one of them played the other one needs to be played. An example is the requirement in a marketplace that product meta data provider is the same as the inventory data provider. CR allows for concurrency where an agent is required to play multiple roles at the same time.
4. **Disjoint Roles (DR):** these are mutually exclusive roles, where only one of them can be played by a system participant at any point of time, and once the system participant plays that role it can not play the other role. An example of this is when you have a Coder and Code reviewer, and the requirement that one can not review his own code (*four eyes principle*).
5. **Amicable Roles (AR):** these are the pairs of roles where the system participant can play one or many of them at the same time without raising any conflict.

The set of all Institutional Roles IR is then represented as: $IR_{msmas} = \langle SR \cup JR \cup CR \cup DR \cup AR \cup HR \rangle$ and each role set in turn is defined as: $R_{inst} = \langle inst, R, R_{rel} \rangle$ where R is the set of roles that belong to institution $inst$ and R_{rel} is the set of relations between these roles in R .

Business Processes. In MSMAS there are two types of business processes models: (i) Composite Business Process (CBP)¹: which is a System Conceptual Plan (SCP) that describes which business processes and/or business activities are needed for the achievement of a Composite System Goal (CSG)²³, and (ii) Basic Business Process (BBP): which contains the detailed specifications of the actual business activities that lead to the achievement of a Basic System Goal (BSG)⁴. Each BSG goal can be achieved through the execution of one or more activities as defined through the system design. MSMAS allows designers to assign any DECLARE-style constraint/relation to any pair of business activities within any BBP⁵. Relations within the context of CBPs however are limited to only four types, as shown in Figure 1 BP/BP relations, where we identify the following types for CBPs as conceptual plans:

1. **Sequential Business Processes (SBP):** these are the pairs of business processes/activities that must be executed consecutively.

¹ Composite Business Process was called in previous publications as Specific Business Process.

² Composite System Goal was called in previous publications as Specific System Goal.

³ Composite System Goal: is a functional goal achievable by one or more business process.

⁴ Basic System Goal: leaf of system goals tree, achievable by one or more business activities.

⁵ All the DECLARE relation formulae, notation and SCIFF mapping appears in [10].

2. **Joint Business Processes (JBP)**: these are pairs of business processes/activities where both are required to be executed, but one after another in a specified order.
3. **Coupled Business Processes (CBP)**: these are pairs of business processes/activities where both are required to be executed, but in no specific order.
4. **Disjoint Business Processes (DBP)**: these are mutually exclusive business processes/activities, where only one may be executed, and once this has occurred the other process/activity can not be executed.
5. **Amicable Business Processes (ABP)**: these are the pairs of business processes/activities that can be executed freely without raising any conflict.

The set of all possible Business Processes/Activities SBP is represented as:

$$SBP_{msmas} = \langle SCP_{msmas} \cup SEP_{msmas} \rangle$$

where the set of all MSMAS System Business Process *CBP* is the set of both System Conceptual Plans *SCP* and the set of System Executable Plans *SEP*:

$$SCP_{msmas} = \langle CBP, SBA, SCP_{rel}, CSG \rangle$$

$$SEP_{msmas} = \langle SBA, SEP_{rel}, BSG \rangle$$

where *SBA* is the set of business activities that lead to the achievement of the System Basic Goals *BSG* and *SEP_{rel}* is the set of relations between these system activities.

Communications Protocols. A communication protocol in MSMAS is a set of one or more messages sent from one system participant to another. We define three types of communication messages:

1. **Inform Message (IM)**: where a system participant sends information in some form such as a belief, a file, etc to another; the sender does not expect a response and the recipient does not expect to reply. This message type is useful for lightweight communications scenarios where acknowledging delivery is not required or essential.
2. **Offer Message (OM)**: where a system participant offers to send some information such as a belief, a file, etc to another system participant. The recipient is expected to respond by accepting or rejecting this offer; if accepted an inform message should follow.
3. **Request Message (RM)**: where a system participant asks for some information from another. A response is required accepting or rejecting the request: either way the response is an inform message or another request message.

In MSMAS a communication protocol *CommuProt* is defined as:

$$CommuProt_{msmas} = \langle Msg_{prot}, Msg_{rel}, IR_{prot} \rangle$$

where a communication protocol *CommuProt_{msmas}* is the set of communication messages *Msg_{prot}* that are exchanged between the system participants playing the institutional *IR_{prot}* roles according to the constraints set by the set of relations *Msg_{rel}* between the pairs of these messages and:

$$Msg_{msmas} = \langle IM \cup OM \cup RM \rangle$$

$$\text{where } msg_{msmas} = \langle sender, recipient, msgContent, timeStamp \rangle$$

3 The SCIFF Framework

SCIFF is a logic programming framework originally proposed by Alberti et al [1]. It is based on Abductive Logic Programming (ALP) [9]. An ALP is a triple $\langle P, A, IC \rangle$,

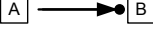
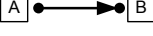
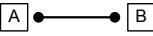
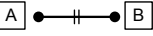

DECLARE Notation	DECLARE Visual notation	MSMAS Role/Role Relation	MSMAS BP/BP Relation
Precedence Relationship: If B is performed A should have been performed before it		Sequential Roles: The system participant has to play Role B only after playing Role A	Sequential BPs: BP B has to be executed only after the execution of BP A
Succession Relationship: every execution of A should be followed by the execution of B and each B should be preceded by A		Joint Roles: The system participant must play Role B after playing Role A , and must have played Role A in order to play Role B	Joint BPs: BP B must be executed after the execution of BP A , and BP A must have been executed to start executing BP B
Coexistence Relationship: If either A or B is performed, the other one has to be executed as well.		Coupled Roles: The system participant has to play both Role A and Role B	Coupled BPs: Both BP A and BP B have to be executed
Not Coexistence Relationship: If one of A or B is performed, the other one can not be executed.		Disjoint Roles: If the system participant plays Role A then Role B can not be played, and vice versa.	Disjoint BPs: If BP A has been executed then BP B can not be executed, and vice versa.
No constraint		Amicable Roles: The system participant can play any or both of Role A and Role B without any restriction	Amicable BPs: Any or both of BP A and BP B can be executed without any restriction as long as their specified pre conditions -if any- are met

Fig. 1. DECLARE Notation and its Mapping to MSMAS Role/Role relation concepts

where A is a set of predicates, named *abducibles*, P is a logic program that uses predicates in A but does not define them, and IC is a set of integrity constraints.

Reasoning in abductive logic programming is a goal-directed task (G , a goal), and amounts to finding an explanation set Δ of (ground) abducible predicates, such that: $P \cup \Delta \models G$ and $P \cup \Delta$ is consistent. The set IC of integrity constraints constrains the explanations Δ for the goal G , through the additional requirement $P \cup \Delta \models IC$.

SCIFF leverages on ALP to constrain the dynamics of an event-based system, such as the interaction between multiple agents [1] or the execution of a business process [12]. In particular, SCIFF instantiates the ALP triple $\langle P, A, IC \rangle$ as follows:

- A is constituted by special predicates denoting expectations about (un)desired events;
- P is a knowledge base used to capture the static knowledge of the targeted system;
- IC is used to relate the occurrence of events to expected events, thus defining which are the events that are expected to occur when a certain trace of BP events is observed.

Events. In SCIFF there is a clear distinction between the description of an event and the occurrence of said event. In fact, an event is represented as a term, whereas an event that has happened is an atom $\mathbf{H}(\text{Event}, \text{Time})$ where *Event* is a *Term* and *Time* is an integer denoting the time at which that event happened. Ground happened events are used to represent a (partial) execution trace of the system, enumerating the relevant events and their timestamps, whereas happened events with variables are used to denote a class of matching ground happened events. For example, $\mathbf{H}(\text{inform}(\text{john}, \text{mary}, \text{call_code}(123)), 5)$ denotes that *john* informed *mary* at time 5 that the call code has value 123. Whereas, $\mathbf{H}(\text{inform}(X, \text{mary}, \text{call_code}(C)), T)$ models that some agent X informed *mary* at a certain time T that the call code has value C .

As well as happened events, SCIFF supports the modelling of (un)desired courses of interaction by introducing the notion of *expected* events, making it possible to explicitly describe what is expected (not) to happen. Expectations can be either positive

$\mathbf{E}(Event, Time)$ or negative $\mathbf{EN}(Event, Time)$. The intuitive quantification for the variables possibly contained in the expectations is existential for positive expectations, and universal for negative expectations. For example, $\mathbf{E}(inform(X, mary, call_code(C)), T)$ models that it is expected that someone informs *mary* about the call code at some point in time, whereas $\mathbf{EN}(inform(X, mary, call_code(C)), T)$ means that no agent can ever inform *mary* about call codes. The full SCIFF event syntax appears in [1].

SCIFF Integrity Constraints. In the SCIFF framework, integrity constraints (*ICs*) are used to express behavioural rules interconnecting happened events with expectations, to represent the expected and forbidden courses of interaction when a given pattern of happened events is found in the current system trace. Technically, they are (forward) implications of the form $\beta(\mathbf{X}) \rightarrow \gamma(\mathbf{X}, \mathbf{Y})$, where $\beta(\mathbf{X})$ is a conjunction of literals, i.e., of (partially grounded) happened/expected events and other predicates, and $\gamma(\mathbf{X}, \mathbf{Y})$ is a disjunction or conjunction of expectations and other predicates. Intuitively, variables \mathbf{X} are universally quantified with the entire implication as scope, whereas variables \mathbf{Y} are existentially or universally quantified depending on whether they appear inside positive or negative expectations (for a full account of quantification, see [1]). Predicates are used to constrain further the matching events, and include Constraint logic programming (CLP)⁶ constraints. When applied to time variables, CLP constraints are particularly useful for imposing metric temporal conditions on happened/expected events. For example, the integrity constraint:

$$\mathbf{H}(create_call(X, C, T) \wedge friend(X, Y) \rightarrow \mathbf{E}(inform(X, Y, call_code(C)), T_2) \wedge T_2 < T + 10$$

states that whenever agent *X* creates a call with code *C*, *X* is expected to inform each of her friends about the value of the call code within 10 time units. Once again, for the full SCIFF social integrity constraint syntax, see [1].

SCIFF Knowledge Base. SCIFF *ICs* can capture the dynamic aspects of a system by interconnecting the observed and expected courses of interaction. However, they are not meant to represent the static knowledge that might be needed to describe the system independently of its dynamics. The SCIFF framework allows the definition of this type of knowledge inside a knowledge base (*KB*). The *KB* can be used to list facts known about the domain under study (such as the extension of the *friend* predicate used in the aforementioned sample integrity constraint), or to encode complex derivation rules modeled as logic programming clauses. Such derivation rules could also employ happened and expected events to provide a-priori definitions for knowledge related to the system dynamics. The full syntax for SCIFF knowledge base terms is given in [1].

Compliance in SCIFF. We now describe the declarative semantics of SCIFF, which builds upon the semantics of ALP and extends it so as to capture the meaning of expectations. In particular, SCIFF declaratively captures the notion of *compliance* of a system execution trace with the modelled specification. This is done by considering positive and negative expectations as abducible predicates, and by introducing the notion of *fulfillment*. Starting from the knowledge base and the set of happened events

⁶ A Constraint Logic Program is a logic program that contains constraints in the body of clauses.

contained in the analyzed trace of the system (which extends the knowledge base with information about the dynamics), expectations are hypothesized consistently with the *ICs* and with an expectation-consistency rule stating that no event can be expected to happen and not to happen at the same time. A positive (respectively negative) expectation is then judged as fulfilled (respectively violated) if there exists a corresponding matching happened event in the trace. This can be considered as a sort of *hypothesis confirmation* step, where the hypothesized courses of execution match with an actual behaviour.

This declarative notion of compliance has an operational counterpart in the SCIFF proof procedure, which concretely realizes an inference mechanism to 1. dynamically acquire happened events reporting about the evolution of the system dynamics, 2. use the modeled knowledge base and integrity constraints so as to generate expectations about the courses of execution, and 3. match expectations with happened events, deciding their fulfillment. Execution traces which fulfill all the generated expectations are then deemed as *compliant* with the specification.

An extension of the SCIFF proof procedure, called *g-SCIFF*, can be used to prove properties of the model at design time, i.e., without having an explicit trace of the system [12]. Given a property, *g-SCIFF* tries to generate a (partially specified) trace showing that the property can be satisfied while respecting all the modeled *ICs*. Intuitively, this is done by transforming every pending positive expectation into a corresponding happened event, checking that no negative expectation becomes violated.

Finally, we observe that while termination of the proof procedures cannot be guaranteed in general, all the techniques developed to check termination of (abductive) logic programs can be seamlessly applied to SCIFF (see, e.g., [12,10] for a discussion on termination conditions when reasoning on extended DECLARE).

4 MSMAS Semantics in SCIFF

In this section we establish a correspondence between MSMAS and SCIFF, consequently enabling the exploitation of the reasoning capabilities of the SCIFF framework to systems modelled with MSMAS. The translation is inspired by [11,10].

4.1 Events in MSMAS

In MSMAS, events reflect the execution of business processes, as well as the dynamics of institutions and of agent interaction. In particular, a system execution is understood by MSMAS in terms of the following events.

Institutional events are triggered when a system participant plays a defined institutional role. For example:

playRole(Agent1, (marketplaceInstitution, seller))

where the agent (*Agent1*) plays the role (*seller*) within the defined institution (*marketplaceInstitution*).

Communications events are triggered when a system participant sends a message within a defined communication protocol. For example:

*requestMsg((buyProductProtocol, customer1, seller5,
getprice(ean : 9782735638938)))*

where the (*customer1*) agent sends request message which is part of (*buyProductProtocol*) communication protocol to get the price of product with ID/EAN (9784431540816) to get the price from (*seller5*) agent.

Business process events are triggered when a system participant starts/ends the execution of an activity within a defined basic business process. For example:

$$\text{startActivity}(\text{checkForNewSSL}(\text{updateFileBP}, (\text{updateAvailabilityCP})), \\ \text{Agent1}, (\text{updateFileBSG}, \text{false}), (\text{timeForUpdate}, \text{true})))$$

where the (*Agent1*) agent starts the execution of (*checkForNewSSL*) activity within the defined basic process (*updateFileBP*) which in turn is a step in the conceptual plan (*updateAvailabilityCP*) to achieve the basic system goal (*updateFileBSG*) with inputs/preconditions (*TimeForUpdate*) with value (*true*). or

$$\text{endActivity}(\text{checkForNewSSL}(\text{updateFileBP}, (\text{updateAvailabilityCP})), \\ \text{Agent1}, (\text{updateFileBSG}, \text{false}), (\text{fileFound}, \text{true})))$$

where the agent *Agent1* ends the execution of activity *checkForNewSSL* within the defined basic process *updateFileBP*, which in turn is a step in the conceptual plan set CP, that contains conceptual plan *updateAvailabilityCP*, to achieve basic system goal *updateFileBSG* with outputs/postconditions *fileFound* with value *true*.

4.2 Institutional Role/Role Relation Formalisation

Given a MSMAS model, each Role/Role relation present in the model is captured as a corresponding fact of the type: $\text{role_role_relation}(A, B, \text{Type})$ For example, $\text{role_role_relation}(\text{Registered_User}, \text{Seller}, \text{sequential_roles})$, expresses that a precedence/sequential role relation holds between *Registered_User* and *Seller* roles. All these facts are grouped together inside an “institution” knowledge base \mathcal{KB}_{inst} .

The Role/Role relations described in Table 1 are then formalized by means of *ICs* that follow the DECLARE to SCIFF translation presented in [11,10]. Such constraints are grouped together into an integrity constraint set \mathcal{IC}_{inst} .

Sequential Roles. A sequential role relation is represented by the following *IC*:

$$\text{role_role_relation}(A, B, \text{sequential_roles}) \\ \wedge \mathbf{H}(\text{play_role}(\text{SystemParticipant}, (\text{Institution}, B), T_B)) \\ \rightarrow \mathbf{E}(\text{play_role}(\text{SystemParticipant}, (\text{Institution}, A), T_A)) \wedge T_A < T_B.$$

Notice that the constraint is instantiated for every Role/Role relation of type *sequential_roles* contained into \mathcal{KB}_{inst} .

Joint Roles can be formalised using the following *ICs*:

$$\text{role_role_relation}(A, B, \text{joint_roles}) \\ \wedge \mathbf{H}(\text{play_role}(\text{SystemParticipant}, (\text{Institution}, A), T_A)) \\ \rightarrow \mathbf{E}(\text{play_role}(\text{SystemParticipant}, (\text{Institution}, B), T_B)) \wedge T_B > T_A. \\ \text{role_role_relation}(A, B, \text{joint_roles}) \\ \wedge \mathbf{H}(\text{play_role}(\text{SystemParticipant}, (\text{Institution}, B), T_B)) \\ \rightarrow \mathbf{E}(\text{play_role}(\text{SystemParticipant}, (\text{Institution}, A), T_A)) \wedge T_A < T_B.$$

Coupled Roles can be captured as joint roles, but without imposing any ordering constraint on the event timestamps:

$$\begin{aligned}
 & \text{role_role_relation}(A, B, \text{coupled_roles}) \\
 & \wedge \mathbf{H}(\text{play_role}(\text{SystemParticipant}, (\text{Institution}, A), T_A)) \\
 & \quad \rightarrow \mathbf{E}(\text{play_role}(\text{SystemParticipant}, (\text{Institution}, B), T_B)). \\
 & \text{role_role_relation}(A, B, \text{coupled_roles}) \\
 & \wedge \mathbf{H}(\text{play_role}(\text{SystemParticipant}, (\text{Institution}, B), T_B)) \\
 & \quad \rightarrow \mathbf{E}(\text{play_role}(\text{SystemParticipant}, (\text{Institution}, A), T_A)).
 \end{aligned}$$

Disjoint Roles are formalized by means of negative expectations:

$$\begin{aligned}
 & \text{role_role_relation}(A, B, \text{disjoint_roles}) \\
 & \wedge \mathbf{H}(\text{play_role}(\text{SystemParticipant}, (\text{Institution}, A), T_A)) \\
 & \quad \rightarrow \mathbf{EN}(\text{play_role}(\text{SystemParticipant}, (\text{Institution}, B), T_B)).
 \end{aligned}$$

4.3 Business Processes and Business Activities Relation Formalisation

In MSMAS, the modelling of Composite Business Processes (CBPs) means representing the conceptual plans broken down into the necessary steps to achieve one specific system goal. Meanwhile the actual executable plans are those that are modelled as Basic Business Processes (BBPs), where the plan steps are business activities (BAs). We allow the system designer to set constraints between BP/BP, BP/BA, and BA/BA at any level, whether part of a conceptual plan or of an executable plan: the only difference is that a BP/BP or BP/BA relationship at the conceptual plan level is inherited all the way down through all sub-processes/sub-plans. BP/BP or BP/BA relations are restricted to the types described in §2 (Sequential Business Processes, Joint Business Processes, Coupled Business Processes, and Disjoint Business Processes) and are formalised in the KB_{SCP} as a fact of the type: $scp_scp_relation(A, B, Type)$. For example, $scp_scp_relation(\text{checkForNewSSL}, \text{publishSupplierUpdate}, \text{sequential_business_process})$, expresses a $sequential_business_process$ relation between the checkForNewSSL and $\text{publishSupplierUpdate}$ business processes.

The formalisation of the relations as integrity constraints follow the same method as for role/role relations in section 4.2. An interesting pattern is the one of $sequential_processes$, which can be represented by means of a DECLARE $chain_response$, in turn captured in SCIFF as follows:

$$\begin{aligned}
 & scp_scp_relation(A, B, \text{sequential_business_process}) \\
 & \wedge \mathbf{H}(\text{execute}(BP_A, \text{SystemParticipant}, \text{Beliefs}), T_A) \\
 & \quad \rightarrow \mathbf{E}(\text{execute}(BP_B, \text{SystemParticipant}, \text{Beliefs}), T_B) \wedge T_B > T_A \\
 & \quad \wedge \mathbf{EN}(\text{execute}(_, _, _), T_x) \wedge T_x > T_A \wedge T_x < T_B.
 \end{aligned}$$

The full set of DECLARE constraints are supported at the BA/BA level, where these business activities are steps within an executable plan. For a comprehensive treatment of such constraints in SCIFF, see [10].

4.4 Communication Protocols Relation Formalisation

Communication protocols serve as a vehicle for enabling the development of interoperable agents and they mainly facilitate negotiation, cooperations and coordination

among all different system participants according to the system design. In MSMAS as explained in Section 2, using the identified three types of messages the system designer can create any custom communication protocol of any length of a finite set of messages. Only Offer Message and Request Message types do require a response when sent, so effectively a request message has a succession relationship with a response message i.e. it is a **Joint Relationship** between two messages that can be formalised in the KB_{MSG} as a fact of the type: $msg_msg_relation(protocol, A, B, Type)$. For example, $msg_msg_relation(getPrices, submitFileUpdate, submitSecurityToken, joint_message)$, expresses a *joint_message* relation between *submitFileUpdate* and *submitSecurityToken* messages in the communication protocol *getPrices*.

The formalisation of the relations as integrity constraints follow the same method as per previous examples in previous sections. **Joint Communication Messages** can be formalised using the following integrity constraints:

$$\begin{aligned}
& msg_msg_relation(prot_1, requestMsg, informMsg, joint_message) \\
& \wedge \mathbf{H}(requestMsg(prot_1, SystemParticipant_A, SystemParticipant_B, Content), T_A) \\
& \rightarrow \mathbf{E}(informMsg(prot_1, SystemParticipant_B, SystemParticipant_A, Content), T_B) \\
& \wedge T_B > T_A. \\
& msg_msg_relation(prot_1, requestMsg, informMsg, joint_message) \\
& \wedge \mathbf{H}(informMsg(prot_1, SystemParticipant_B, SystemParticipant_A, Content), T_B) \\
& \rightarrow \mathbf{E}(requestMsg(prot_1, SystemParticipant_A, SystemParticipant_B, Content), T_A) \\
& \wedge T_A < T_B.
\end{aligned}$$

4.5 Reasoning about MSMAS Models

By putting together the translation principles presented above, we obtain a full SCIFF specification constructed as follows:

$$\begin{aligned}
& \mathcal{P}_{msmas} \equiv \langle KB_{msmas}, \{\mathbf{E}/2, \mathbf{EN}/2\}, \mathcal{IC}_{msmas} \rangle \\
\text{where } & KB_{msmas} \triangleq KB_{inst} \cup KB_{prot} \cup KB_{act} \\
\text{and } & \mathcal{IC}_{msmas} \triangleq \mathcal{IC}_{inst} \cup \mathcal{IC}_{prot} \cup \mathcal{IC}_{act}
\end{aligned}$$

The SCIFF and g-SCIFF proof procedure can consequently be applied to reason about MSMAS models. Notably, the joint application of these proof procedures covers the entire lifecycle of a system: at design time, the SCIFF proof procedure can be used to check compliance of simulated event traces, while g-SCIFF can be applied to verify whether the MSMAS model of the system meets some desired properties; at runtime, the SCIFF proof procedure can be employed to monitor the running system and check whether it fulfils the generated expectations; a-posteriori, the same approach can be employed to analyze complete traces representing past system executions.

More specifically, the correspondence between MSMAS and SCIFF gives an expectation-based declarative semantic for MSMAS, providing a formal notion of *compliance* between an execution trace of the system (also called *instance* of the specification) and the constraints obtained from the MSMAS model. Intuitively, given a set **HAP** of happened events, \mathcal{P}_{msmas} leads to formulate an abductive set **EXP** that contain positive and negative expectations, which reflect the events that are expected (not) to occur in the state of affairs obtained after the execution of the events in **HAP**. In this

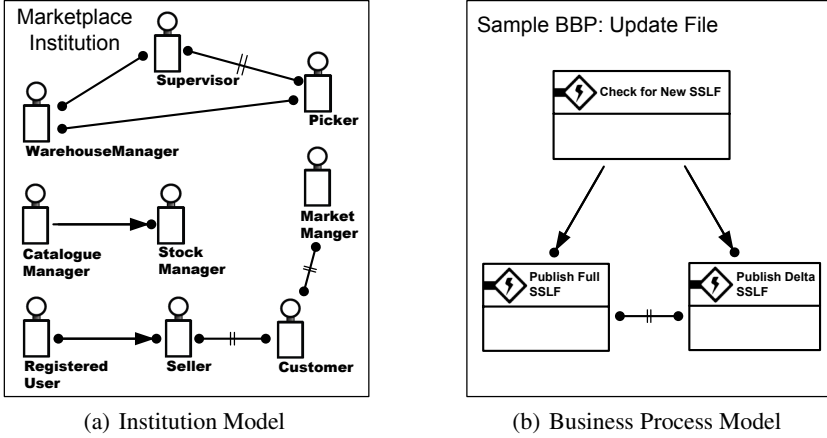


Fig. 2. MSMAS example models

respect, we take advantage of the declarative semantics of SCIFF to tackle three basic reasoning tasks: *consistency*, *fulfillment*, and *conformance*.

Consistency states that a MSMAS event cannot be expected to happen and expected not to happen at the same time. Technically, for each (ground) MSMAS event e and timestamp t , consistency requires that $\{\mathbf{E}(e, t), \mathbf{EN}(e, t)\} \not\subseteq \mathbf{EXP}$. Notice that consistency is not checked by the proof procedures by effectively grounding the expectations, but by using variables and CLP constraints to maintain an intensional, “symbolic” representation of (classes of) expectations, using constraint-solving to detect clashes between positive and negative expectations.

Fulfillment expresses the semantics of expectations. In particular, we say that a positive expectation $\mathbf{E}(e, t) \in \mathbf{EXP}$ is fulfilled by a set \mathbf{HAP} of happened events if and only if $\mathbf{H}(e, t) \in \mathbf{HAP}$, i.e., a corresponding happened event has occurred. Specifically, a negative expectation $\mathbf{EN}(e, t) \in \mathbf{EXP}$ is fulfilled by a set \mathbf{HAP} of happened events if $\mathbf{H}(e, t) \notin \mathbf{HAP}$, i.e., no corresponding happened event has occurred. Furthermore, we say that \mathbf{EXP} is fulfilled by \mathbf{HAP} if every expectation in \mathbf{EXP} is fulfilled by \mathbf{HAP} .

Conformance combines the notion of consistency and fulfillment to characterize whether a trace of the system respects all the constraints imposed by the MSMAS model. In particular, given a goal G and a complete trace of the system \mathbf{HAP} , we say that \mathbf{HAP} *conforms to* the MSMAS model satisfying G if:

$$\begin{aligned}
 \mathcal{KB}_{msmas} \cup \mathbf{HAP} \cup \mathbf{EXP} &\models G \\
 \mathcal{KB}_{msmas} \cup \mathbf{HAP} \cup \mathbf{EXP} &\models \mathcal{IC}_{msmas} \\
 &\mathbf{EXP} \text{ is consistent} \\
 &\mathbf{EXP} \text{ is fulfilled by } \mathbf{HAP}
 \end{aligned}$$

Looking closely at the sample institution model in Figure 2, we observe that the specifications of *WarehouseManager*, *Supervisor* and *Picker* eventually lead to an inconsistency as soon as an agent start to play one of these roles: *WarehouseManager* and *Supervisor* are joint roles, and so are *WarehouseManager* and *Picker*, which implies that also *Supervisor* and *Picker* have to be joint roles, while the model constrains them

to be disjoint. This inconsistency can be detected by the *SCIFF* proof procedure when a partial trace of the system is analyzed. With *g-SCIFF*, instead, the problem can, e.g., be detected when the modeller asks the following query: *is it possible for an agent to play the role of Picker?* Observe that a negative answer to this query would seriously question the correctness of the *MSMAS* model, because it would attest that *Picker* is always an “empty” role in any possible execution. In *g-SCIFF*, such a query can be expressed in terms of the goal: $\mathbf{E}(\text{play_role}(SP, (I, picker), T))$. To answer this query, *g-SCIFF* tries to generate a (partially specified) trace that conforms to the *MSMAS* model and at the same time satisfies the goal. In particular, starting from the expectation mentioned in the goal, the proof procedure generates a happened event that fulfils the expectation: $\mathbf{H}(\text{play_role}(SP, (I, picker), T))$. This event states that at some time T , an agent SP will indeed play the *Picker* role in the context of some institution I . Due to the two constraints attached to the *Picker* role in Figure 2(a), this in turn triggers the generation of two expectations: one stating that the *WarehouseManager* role must also be played by that agent ($\mathbf{E}(\text{play_role}(SP, (I, warehouse_manager), T_2))$), and the other stating that the *Supervisor* role can never be played by that agent ($\mathbf{EN}(\text{play_role}(SP, (I, supervisor), T_3))$, where T_3 is universally quantified). To fulfill the first expectation, *g-SCIFF* generates a corresponding happened event, which however triggers a further positive expectation about the fact that the agent must also play, sooner or later, the *Supervisor* role ($\mathbf{E}(\text{play_role}(SP, (I, supervisor), T_4))$, where T_4 is existentially quantified). This expectation clashes with the negative expectation generated before, leading to inconsistency and, in turn, to a negative answer for the posed query.

5 Discussion and Future Work

As far as this presentation is concerned, we do make some simplifying assumptions about scenarios being modelled of which we highlight: (i) all system goals are compatible in that they lead to the achievement of the general goal of the system. The designer can add conflicting goals if the intention is to construct a competitive, however *MSMAS* and its tool does not currently support the identification of conflicting system goals and does not provide any guidance as yet, on how to resolve such issues. (ii) it is the designer’s responsibility to specify how to manage the registration and deregistration of system participants with respect to an institution. The registration process is normally used as a mechanism for declaring the role(s) that a system participant intends to play and by so doing, the institution governor may validate the participant’s eligibility, prior to starting any activity with that role.

Meta-models for norm-aware MAS are an active research area as an incomplete selection from the literature demonstrates. The *Alive* project [2] used model-driven development to deploy agents and services in the context of an organizational model defined in *Opera*, which expresses norms in terms of states to be maintained or avoided. However, *Opera* lacks an activity model, making run-time validation difficult. The very detailed formalisation of *EIDE* [3] takes what is arguably a more low-level approach to capturing the quite complex semantics, particularly those associated with scenes and transitions, using the *Z* specification language. While this provides a high degree of precision, there is no associated meta-model, so although there are tools to generate code automatically, they do not have that formal backing. A further issue is the relatively stylised nature of *EIDE* agents, whose actions have historically been highly

constrained through the use of governors that ensure non-compliant actions never occur. Ghorbani [8] provides a different perspective, in which a meta-model is defined and used for model-driven development in the context of agent-based simulations, informed by Ostrom's IAD framework. While this emphasizes the role of norm and institution in the governance of agent behaviour, the meta-model reflects the research objectives of modelling social structures and their evolution. Against this background – and much literature for which there is not room here – we suggest that (i) the novel combination of a business-oriented institutional meta-model, designed to support self-management in use, (ii) with a design- and run-time formal proof mechanism, (iii) provides a new perspective on formal software engineering of MAS and contributes towards the evolution of and the debate on the application of meta-models and model-driven development in MAS. Future plans for MSMAS include the enhancement of its metamodel to be Model Driven Development (MDD) compliant and to investigate and include the modelling of multiple organisations/institution interactions.

References

1. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: The sciff framework. *ACM Trans. Comput. Logic* 9(4), 29:1–29:43 (2008)
2. Aldewereld, H., Padget, J., Vasconcelos, W., Vázquez-Salceda, J., Sergeant, P., Staikopoulos, A.: Adaptable, Organization-Aware, Service-Oriented Computing. *IEEE Intelligent Systems* 25(4), 26–35 (2010)
3. d'Inverno, M., Luck, M., Noriega, P., Rodríguez-Aguilar, J.A., Sierra, C.: Communicating open systems. *Artificial Intelligence* 186, 38–64 (2012)
4. Baldoni, M., Baroglio, C., Mascardi, V., Omicini, A., Torroni, P.: Agents, Multi-Agent Systems and Declarative Programming: What, When, Where, Why, Who, How? In: Dovier, A., Pontelli, E. (eds.) *GULP. LNCS*, vol. 6125, pp. 204–230. Springer, Heidelberg (2010)
5. Elakehal, E.E., Padget, J.: Msmas: Modelling self-managing multi agent systems. *SCPE: Scalable Computing: Practice and Experience* 13(2), 121–137 (2012)
6. Elakehal, E.E., Padget, J.: A practical method for developing multi agent systems: APMD-MAS. In: Brazier, F.M.T., Nieuwenhuis, K., Pavlin, G., Warnier, M., Badica, C. (eds.) *Intelligent Distributed Computing V. SCI*, vol. 382, pp. 11–20. Springer, Heidelberg (2011)
7. Alberola, J.M., Such, J.M., Botti, V., Espinosa, A., Garcia-Fornes, A.: A scalable multiagent platform for large systems. *Computer Science and Information Systems* 10(1), 51–77 (2013)
8. Ghorbani, A., Bots, P., Dignum, V., Dijkema, G.: Maia: a framework for developing agent-based social simulations. *Journal of Artificial Societies and Social Simulation* 16(2), 9 (2013)
9. Kakas, A.C., Kowalski, R.A., Toni, F.: *Abductive logic programming* (1993)
10. Montali, M. (ed.): *Specification and Verification of Declarative Open Interaction Models. LNBIP*, vol. 56. Springer, Heidelberg (2010)
11. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographiess. *ACM Trans. Web* 4(1), 3:1–3:62 (2010)
12. Montali, M., Torroni, P., Chesani, F., Mello, P., Alberti, M., Lamma, E.: Abductive logic programming as an effective technology for the static verification of declarative business processes. *Fundamenta Informaticae* 102(3-4), 325–361 (2010)

13. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) *BPM Workshops 2006*. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006)
14. van der Aalst, W., Pesic, M.: Decserflow: Towards a truly declarative service flow language. In: Leymann, F., et al. (eds.) *The Role of Business Processes in Service Oriented Architectures*, Dagstuhl, Germany, vol. 06291, Dagstuhl Seminar Proceedings (2006)
15. Zambonelli, F., Jennings, N.R., Wooldridge, M.: *Organisational rules as an abstraction for the analysis and design of multi-agent systems* (2001)