Verification of GSM-based Artifact-centric Systems through Finite Abstraction

Francesco Belardinelli¹, Alessio Lomuscio¹, and Fabio Patrizi²

¹ Department of Computing, Imperial College London {f.belardinelli,a.lomuscio}@imperial.ac.uk ² DIIAG, Sapienza Università di Roma patrizi@dis.uniromal.it

Abstract. The GSM framework provides a methodology for the development of artifact-centric systems, an increasingly popular paradigm in service-oriented computing. In this paper we tackle the problem of verifying GSM programs in a multi-agent system setting. We provide an embedding from GSM into a suitable multi-agent systems semantics for reasoning about knowledge and time at the first-order level. While we observe that GSM programs generate infinite models, we isolate a large class of "amenable" systems, which we show admit finite abstractions and are therefore verifiable through model checking. We illustrate the contribution with a procurement use-case taken from the relevant business process literature.

1 Introduction

The Artifact-centric paradigm [6,7] has recently gained considerable prominence in the business processes and services communities as a promising and novel methodology for quick and inexpensive deployment of data-intensive web-services. In the artifact-centric approach *data* feature prominently and drive the execution of the system, together with the associated process-based description of the services. The Guard-Stage-Milestone (GSM) language [10], together with its Barcelona production and execution suite, provides a declarative framework to deploy artifact-centric systems. In a nutshell, GSM offers the constructs for the definition of *artifacts* as typed records of data, their evolution (or *lifecycles*) through a dedicated rule-driven semantics, and the interface for the interaction of the artifact-systems with possible users. Such interface is in fact composed of services, i.e., operations (or *tasks*), each with its interface and semantics, that external actors can invoke, which affect the current state of the artifacts in the system, and may in turn trigger the execution of other operations.

In this paper we tackle the topic above from a verification perspective. We see two considerable deficiencies in the GSM approach as it currently stands. Firstly, similarly to the database-inspired techniques, GSM programs only define the evolution of the artifact-system and provide no precise mechanism for accounting for any users or automatic agents interacting with the system. Yet, if we wish to follow an approach of implementing services through agents, these need to be present in the model. Secondly, GSM currently lacks any support for automatic verification. Yet, validation through verification is increasingly being regarded as an important aspect of service deployment [12, 15, 17].

This paper aims to make a direct contribution towards these two key problems. To solve the first concern, we provide a semantics, based on multi-agent systems, to GSM programs where we give first-class citizenship to human actors and automatic agents present in the service composition. To solve the second, we observe that GSM programs generate infinite-state systems thereby making traditional model checking impracticable. Our key contribution here is to show that GSM programs admit, under the rather general conditions we identify below, finite models, thereby opening the way for their effective verification. We show through one example that the finite models required are indeed of manageable size. The rest of the paper is organised as follows. In Section 2 we introduce artifacts and GSM programs, which are illustrated by the RPO scenario in Section 3. In Section 4 we adopt the semantics of AC-MAS to deal with the verification of GSM programs. Finally, in Section 5 we show how to embed GSM programs into AC-MAS; thus obtaining finite abstractions for the former.

Related work. The exploration of finite abstraction in the context of artifact-centric environments has attracted considerable attention recently [2–5, 8, 9, 14]. While all works cited make a noteworthy contribution and are in some cases used here as a stepping stone for our results [2–4], the key point of departure from the literature of the present contribution is that we here operate directly on GSM programs and not on logical models derived manually from them. We see this as an essential step towards the construction of *automatic* verification techniques for GSM programs.

2 GSM Programs

Artifact-centric systems are based on the notion of *artifact*, or a record of structured data, that are born, evolve, and die during a system run either as a consequence of chains of internal actions of other artifacts, or through external actions performed by actors. GSM [10] is a declarative language, interpreted by specialised toolkits, that enables the user to implement sophisticated guard-stage-milestone models for artifact systems.

For simplicity, here we work on an untyped version of GSM programs in which we also neglect timestamps: while GSM programs are richer, the version we consider enables us to present concisely our decidability results while at the same time supporting complex use-cases as we show in Section 3. The present section makes use of notions and definitions from [10].

Definition 1 (Artifact Type). An artifact type is a tuple $AT = \langle P, x, Att, Stg, Mst, Lcyc \rangle$ s.t.

- *P* is the name of the artifact type;
- x is a variable that ranges over the IDs of instances of AT; this is the context variable of AT, which is used in the logical formulas in Lcyc;
- Att is the set of attributes, which is partitioned into the set Att_{data} of data attributes and Att_{status} of status attributes;
- *Stg is the set of* stages;
- *Mst is the set of* milestone;
- Lcyc is the lifecycle model of the artifact type AT, which is formally defined below.

Intuitively, artifact types can be seen as records of structured data. Stages, milestones and lifecycles will be introduced shortly. Also, Att_{data} includes the attribute mostRecEventType, which holds the type of the most recent event. Milestones and stages describe the evolution of the artifact type. For each milestone $m \in Mst$, in Att_{status} there is a boolean *milestone status* attribute, denoted as m. Analogously, for each stage $S \in Stg$, in Att_{status} there is a boolean stage status attribute, $active_S$.

While the data content of an artifact type is specified by its datamodel, i.e., all its attributes excluding the lifecycle, its execution is described by its lifecycle.

Definition 2 (Lifecycle). *The lifecycle of an artifact type AT is a tuple* $Lcyc = \langle Substg, Task, Owns, Guards, Ach, Inv \rangle$ s.t.

- Substg is a function from Stg to finite subsets of Stg, where the relation $\{(S, S')|S' \in Substg(S)\}$ is a forest. The leaves of this forest are called atomic stages.
- *Task is a function from the atomic stages in Stg to tasks.*
- Owns is a function from Stg to finite, non-empty subsets of Mst. A stage S owns a milestone m if $m \in Owns(S)$.
- Guards is a function from Stg to finite sets of sentries, as defined in Section 2.1. An element of Guards(S) is called a guard for S.
- Ach is a function from Mst to finite sets of achieving sentries.
- Inv is a function from Mst to finite sets of invalidating sentries.

We will give all details on the notions above in Section 2.1. Intuitively, every artifact goes through a number of stages, which are activated by the relevant guards. A stage is closed when the tasks associated with it and its substages are fulfilled. When this happens, the milestones associated with the stage become true and possibly trigger the guards associated with another stage.

We now introduce GSM programs.

Definition 3 (GSM program). A GSM program Γ is a set of artifact types AT_i for $i \leq n$.

As a convenience, we assume that all context variables are distinct. Artifact instances are then defined as mappings from artifact types to a possibly infinite interpretation domain U of data values.

Definition 4 (AT and GSM snapshot). A snapshot for the artifact type AT is a mapping σ from x, Att to the interpretation domain U.

A snapshot for the GSM program Γ is a mapping Σ from each type $AT \in \Gamma$ to a set $\Sigma(AT)$ of snapshots for type AT.

Intuitively, a snapshot for the artifact type AT is an assignment of the values in U to the attributes of AT. A GSM snapshot is then a collection of AT snapshots. Notice that different instances of the same artifact type are distinguished by their IDs $\sigma(x)$, hereafter denoted as ρ .

2.1 Execution of GSM Programs

In GSM programs events are responsible for the evolution of the system from one snapshot to the next. Three types of incoming events are considered: 1-way messages M, 2-way service call returns F^{return} , and artifact instance creation requests $create_{AT}^{call}$. A ground event e has a payload $(A_1 : c_1, \ldots, A_n : c_n)$ where A_i is a data attribute and c_i is a value in the domain U. Intuitively, incoming events are processed by the sentries associated with guards and milestones, and the payload determines the evolution of the stage as detailed below.

Definition 5 (**Immediate effect**). The immediate effect of a ground event e on a snapshot Σ , or ImmEffect(Σ , e), is the snapshot that results from incorporating e into Σ , including (i) changing the values of the mostRecEventType attribute of affected (or created) artifact instances; (ii) changing the values of data attributes of affected artifact instances, as indicated by the payload of e.

The operational semantics for GSM programs is built around the notion of *business* step, or B-step, which represents the impact of a single ground incoming event e on a snapshot Σ . The semantics of B-steps is characterised by 3-tuples (Σ, e, Σ') s.t.

- 1. Σ is the *previous* snapshot;
- 2. e is a ground incoming event;
- 3. Σ' is the *next* snapshot;
- 4. there is a sequence $\Sigma_0, \Sigma_1, \ldots, \Sigma_n$ of snapshots s.t. (i) $\Sigma_0 = \Sigma$; (ii) $\Sigma_1 = ImmEffect(\Sigma, e)$; (iii) $\Sigma_n = \Sigma'$; and (iv) for $1 \le j \le n 1$, Σ_{j+1} is obtained from Σ_j by a PAC rule.

Business steps involve *Prerequisite-Antecedent-Consequent* (PAC) rules. To introduce PAC rules we first define formally event expressions and sentries. In what follows τ_{AT} is a path expression x. < path > where x is the ID variable for some artifact type $AT \in \Gamma$. An example of a path expression is $\rho.S.m$, which refers to the milestone m of stage S, for some AT instance ρ .

Definition 6 (Event expression). An event expression $\xi(x)$ for an artifact type AT with ID variable x has one of the following forms:

- Incoming event expression x.e: (i) x.M for 1-way message type M; (ii) x. F^{return} for service call return from F; (iii) x.create^{call}_{AT} for a call to create an artifact instance of type AT.
- Internal event expression: (i) $+\tau_{AT'}$.m and $-\tau_{AT'}$.m, where m is a milestone for type AT'; (ii) $+\tau_{AT'}$.actives and $-\tau_{AT'}$.actives, where S is a stage of type AT'.

Intuitively, an event occurrence of type $+\tau_{AT'}.m$ (resp. $-\tau_{AT'}.m$) arises whenever the milestone m of the instance identified by x. < path > changes value from false to true (resp. true to false). Similarly, an event occurrence of type $+\tau_{AT'}.active_S$ (resp. $-\tau_{AT'}.active_S$) arises whenever the stage S of the instance identified by x. < path > changes value from closed to open (resp. open to closed).

We can now define sentries for guards and milestones. These represent the conditions to open and close stages. **Definition 7** (Sentry). A sentry for an artifact type AT is an expression $\chi(x)$ having one of the following forms: on $\xi(x)$ if $\varphi(x) \wedge x$. active_S, on $\xi(x)$, or if $\varphi(x) \wedge x$. active_S s.t. (i) $\xi(x)$ is an event expression; and (ii) $\varphi(x)$ is an FO-formula over the artifact types occurring in Γ that has exactly one free variable.

We now discuss the interpretation of sentries, i.e., when a snapshot Σ satisfies a sentry χ , or $\Sigma \models \chi$. Satisfaction of an FO-formula φ at Σ is defined as standard. Further, the expression $\rho.e$ for an artifact instance ρ is true at Σ if $\rho.mostRecEventType = e$. Finally, the internal event expression $\odot\rho.\tau.s$ for polarity $\odot \in \{+, -\}$, path expression τ , and status attribute *s*, is true at Σ if the value of $\rho.\tau.s$ matches the polarity.

We can now introduce PAC rules.

Definition 8 (PAC rules). A PAC rule is a tuple $\langle \pi(x), \alpha(x), \gamma(x) \rangle$ s.t.

- $\pi(x)$ is of the form $\tau.m$, $\neg \tau.m$, $\tau.active_S$, or $\neg \tau.active_S$;
- $\alpha(x)$ is of the form $\chi(x) \land \psi(x)$, where $\chi(x)$ is a sentry and $\psi(x)$ is of the form $\tau.m, \neg \tau.m, \tau.active_S, or \neg \tau.active_S;$
- $\gamma(x)$ is an internal event expression as in Def. 6.

Given a B-step $\Sigma = \Sigma_0, \Sigma_1, \ldots, \Sigma_j$ for a ground event *e*, the PAC rule $\langle \pi, \alpha, \gamma \rangle$ is applicable if $\Sigma \models \pi$ and $\Sigma_j \models \alpha$. Applying such a rule yields a new snapshot Σ_{j+1} , which is constructed from Σ_j by applying the effect called for by γ .

Additional conditions can also be assumed for the application of PAC rules, which notably ensure the absence of cycles. We do not present these here and refer to [10] for further information.

3 The RPO Scenario

The *Requisition and Procurement Orders* (RPO) scenario is a business process use-case in which a GSM program is used to implement the procurement process in a business setting [10]. We illustrate the notions presented in Section 2 in the context of a fragment of this scenario. In RPO a *Requisition Order* is sent by a Customer to a Manufacturer to request some goods or services. The Requisition Order has one or more *Line Items*, which are bundled into *Procurement Orders* and sent to different Suppliers. A Supplier can either accept or reject a Procurement Order. In the latter case, the rejected Line Items are bundled into new Procurement Orders. Otherwise, the order is fulfilled by the Supplier and sent to the Manufacturer, who in turn forwards it to the Customer.

In GSM programs it is natural to model the Requisition and Procurement Orders as artifact types *RO* and *PO* respectively. In particular, the *datamodel* of the Requisition Order, i.e., all its attributes excluding the lifecycle, can be encoded as in Fig. 1, which is adapted from [10]. The definition of the Procurement Order datamodel is similar. Notice that in the datamodel we have both data and status attributes; the latter contain milestone and stage data as detailed in Def. 1.

Further, Fig. 2 illustrates part of the lifecycle for the Requisition Order [10]. Stages are represented as rounded boxes. The stage Create Proc Orders contains the child-stages Launching Line Items and Planning Proc Orders; the former is atomic. Mile-stones are shown as small circles associated with stages. For instance, the milestones All



Fig. 1. The Requisition Order datamodel.



Fig. 2. A stage of the Requisition Order lifecycle.

Line items ordered and Req Order cancelled are associated with the stage Create Proc Orders. The former is an achieving milestone, i.e., when All Line items ordered becomes true the stage is closed; while the latter is invalidating, that is, when Req Order cancelled holds, the stage is reopened. The diamond nodes are guards. The stage Create Proc Orders is triggered by guards Initiate Req Order and Re-order Line Items of Rejected Proc Orders. A diamond with a cross represents a "bootstrapping" guard, which indicates the conditions to create new artifact instances. Similar representations can be given for all other stages in the Requisition and Procurement Orders.

As discussed in Section 2 the execution of GSM programs is governed by PAC rules. To illustrate these we consider PAC2 as given in [10]:

	Prerequisite π	Antecedent α	Consequent γ
PAC2	$x.active_S$	on $e(x)$ if $\varphi(x)$	+x.m

where stage S has milestone m and on e(x) if $\varphi(x)$ is an achieving sentry for m. Suppose that $\Sigma_0, \Sigma_1, \ldots, \Sigma_j$ is a sequence of snapshots in a B-step. Intuitively, if $\Sigma \models \pi$ then there is an artifact instance ρ s.t. ρ . $active_S$ is true, i.e., the stage S is active for ρ . Furthermore, if $\Sigma_j \models \alpha$ then ρ . mostRecEventType = e and the achieving condition φ for milestone m holds. Finally, Σ_{j+1} is obtained by applying $+\rho.m$, i.e., by toggling true the flag m for the milestone status of S.

As a result, we clearly see that GSM programs are expressive enough to formalise business processes such as the Requisition and Procurement Orders scenario.

4 Artifact-Centric MAS with Parametric Actions

In Section 5 we introduce a sufficient condition for obtaining finite abstractions for a notable class of the GSM programs. In order to define an embedding into an agent-

based semantics, as well as to state precisely the model checking problem for these structures, we here begin our contribution by generalising the framework of [4] to parametric actions. As we will see later, this is required to obtain effective model checking procedures. While the material extends [4], it follows its structure and some of the definitions.

We start by introducing some terminology on databases [1].

Definition 9 (Database schema and instance). A database schema *is a set* $\mathcal{D} = \{P_1/q_1, \dots, P_n/q_n\}$ of predicate symbols P_i with arity $q_i \in \mathbb{N}$.

A \mathcal{D} -instance on a (possibly infinite) domain U is a mapping D associating each predicate symbol P_i with a finite q_i -ary relation over U, i.e., $D(P_i) \subseteq U^{q_i}$.

The set $\mathcal{D}(U)$ denotes all \mathcal{D} -instances on the domain U. The active domain ad(D) of D is the finite set of all individuals occurring in some predicate interpretation $D(P_i)$. The primed version of a database schema \mathcal{D} as above is the schema $\mathcal{D}' = \{P'_1/q_1, \ldots, P'_n/q_n\}$. Given two \mathcal{D} -instances D and D', $D \oplus D'$ is the $(\mathcal{D} \cup \mathcal{D}')$ -instance s.t. (i) $D \oplus D'(P_i) = D(P_i)$; and (ii) $D \oplus D'(P'_i) = D'(P_i)$. The \oplus operator will be used later in relation with temporal transitions in artifact systems.

We now extend the definition of AC-MAS in [4] to accommodate parametric actions, where U is the interpretation domain.

Definition 10 (Agent). An agent is a tuple $i = \langle D_i, L_i, Act_i, Pr_i \rangle$ s.t.

- \mathcal{D}_i is the local database schema;
- $L_i \subseteq \mathcal{D}_i(U)$ is the set of local states l_i ;
- Act_i is the set of local actions $\alpha_i(\vec{x})$ with parameters \vec{x} ;
- $Pr_i: L_i \mapsto 2^{Act_i(U)}$ is the local protocol function, where $Act_i(U)$ is the set of ground actions $\alpha_i(\vec{u})$ for $\vec{u} \in U^{|\vec{x}|}$.

Given a set $Ag = \{0, ..., n\}$ of agents, we define the *global* database schema of Ag as $\mathcal{D} = \mathcal{D}_0 \cup \cdots \cup \mathcal{D}_n$, i.e., the set of all predicate symbols appearing in some local database schema. Also, agent 0 is usually referred to as the *environment* E.

Agents can be thought of as modules that can be composed together to obtain AC-MAS.

Definition 11 (AC-MAS). Given a set Ag of agents, an artifact-centric multi-agent system is a tuple $\mathcal{P} = \langle S, U, s_0, \tau \rangle$ s.t.

- $S \subseteq L_E \times L_1 \times \cdots \times L_n$ is the set of reachable global states;
- *U* is the interpretation domain;
- $s_0 \in S$ is the initial global state;
- τ : $S \times Act(U) \mapsto 2^S$ is the transition function, where $Act = Act_E \times Act_1 \times \cdots \times Act_n$ is the set of global actions, Act(U) is the set of ground actions, and $\tau(\langle l_E, l_1, \ldots, l_n \rangle, \vec{\alpha}(\vec{u}))$ is defined iff $\alpha_i(\vec{u}) \in Pr_i(l_i)$ for every $i \in Ag$.

In the rest of the paper we assume the function τ to be computable. In particular, for any state $s \in S$ and ground action $\alpha(\vec{u}) \in Act(U)$, the set $|\tau(s, \alpha(\vec{u}))|$ of successors of s is finite. This allows us to abstract from the way τ is actually defined. As it will become apparent in Section 5, GSM programs induce a τ that fulfills this requirement. Also, we assume that no confusion arises with path expressions in GSM programs.

We can interpret a global state $s = \langle l_E, l_1, \ldots, l_n \rangle$ as the \mathcal{D} -instance D s.t. $D(P_i) = \bigcup_{j \in Ag} l_j(P_i)$, for $P_i \in \mathcal{D}$. Notice that for each $s \in S$ there exists a unique \mathcal{D} -instance D as above, however the converse is not true in general. The way D has to be interpreted will be clear from the context. We define the *transition relation* $s \to s'$ if there exists $\vec{\alpha}(\vec{u}) \in Act(U)$ and $s' \in \tau(s, \vec{\alpha}(\vec{u}))$. The notion of *reachability* is defined as in [4]. In what follows we assume that the relation \to is serial, and that S is the set of states reachable from s_0 . Notice that by definition S is infinite in general. Hence, the AC-MAS \mathcal{P} is an infinite-state system. Finally, s and s' are *epistemically indistinguishable* for agent i, or $s \sim_i s'$, if $l_i(s) = l_i(s')$. This is consistent with the standard definition of knowledge as identity of local states [11].

As we are working in a multi-agent system setting, we require to specify temporalepistemic properties of the system and be able to quantify on individuals.

Definition 12 (FO-CTLK). Given a set Var of individual variables and a set $Con \subseteq U$ of individual constants, the FO-CTLK formulas φ on the database schema D are defined in BNF as follows:

$$\varphi ::= t = t' \mid P_i(\vec{t}) \mid \neg \varphi \mid \varphi \to \varphi \mid \forall x \varphi \mid A X \varphi \mid A \varphi U \varphi \mid E \varphi U \varphi \mid K_i \varphi$$

where $P_i \in D$, \vec{t} is a q_i -tuple of terms, and t, t' are terms, i.e., elements in $Var \cup Con$.

The language FO-CTLK is the extension to first-order of the branching time logic CTL enriched with an epistemic operator K_i for each agent $i \in Ag$ [2]. For a formula φ we denote the set of variables as $var(\varphi)$, the set of free variables as $fr(\varphi)$, and the set of constants as $con(\varphi)$. We consider also the non-modal first-order fragment of FO-CTLK, obtained by omitting the modal operators in Def. 12.

An assignment is a function $\sigma : Var \mapsto U$. We denote by $\sigma \begin{pmatrix} x \\ u \end{pmatrix}$ the assignment s.t. (i) $\sigma \begin{pmatrix} x \\ u \end{pmatrix}(x) = u$; and (ii) $\sigma \begin{pmatrix} x \\ u \end{pmatrix}(x') = \sigma(x')$ for $x' \neq x$. We assume that no confusion will arise between assignments in AC-MAS and snapshots in GSM programs. Also, we assume a Herbrand interpretation of constants.

Definition 13 (Semantics of FO-CTLK). We define whether an AC-MAS \mathcal{P} satisfies a formula φ in a state s under assignment σ as standard [4]. In particular,

 $\begin{array}{ll} (\mathcal{P},s,\sigma) \models P_i(t_1,\ldots,t_{q_i}) & \textit{iff } \langle \sigma(t_1),\ldots,\sigma(t_{q_i}) \rangle \in s(P_i) \\ (\mathcal{P},s,\sigma) \models t = t' & \textit{iff } \sigma(t) = \sigma(t') \\ (\mathcal{P},s,\sigma) \models \forall x \varphi & \textit{iff for all } u \in ad(s), \ (\mathcal{P},s,\sigma\binom{x}{u}) \models \varphi \\ (\mathcal{P},s,\sigma) \models K_i \varphi & \textit{iff for all } s', \ s \sim_i s' \textit{ implies } (\mathcal{P},s',\sigma) \models \varphi \end{array}$

A formula φ is true at s, written $(\mathcal{P}, s) \models \varphi$, if $(\mathcal{P}, s, \sigma) \models \varphi$ for all assignments σ ; φ is true in \mathcal{P} , written $\mathcal{P} \models \varphi$, if $(\mathcal{P}, s_0) \models \varphi$.

Note that we adopt an *active domain* semantics, that is, quantified variables range over the active domain of *s*.

Given an AC-MAS \mathcal{P} and an FO-CTLK formula φ , the *model checking problem* amounts to finding an assignment σ such that $(\mathcal{P}, s_0, \sigma) \models \varphi$. Since \mathcal{P} has infinitely many states, the corresponding model checking problem is undecidable in general. However, in Section 5.1 we show that there are notable exceptions.

4.1 Finite Abstractions

In this section we extend the techniques in [4] to define finite abstractions for AC-MAS with parametric actions. We omit proofs for reasons of space. We fix a finite set $C \supseteq ad(s_0)$ of constants. Further, whenever we consider an FO-CTLK formula φ , we assume w.l.o.g. that $con(\varphi) \subseteq C$. Finally, the states s and s' are defined on the interpretation domains U and U' respectively, and $\mathcal{P} = \langle S, U, s_0, \tau \rangle$ and $\mathcal{P}' = \langle S', U', s'_0, \tau' \rangle$ are AC-MAS.

To introduce the notion of *bisimulation* as defined in [4], we first need to state when two states are *isomorphic*.

Definition 14 (Isomorphism). The states s and s' are isomorphic, or $s \simeq s'$, iff there exists a bijection $\iota : ad(s) \cup C \mapsto ad(s') \cup C$ s.t. (i) ι is the identity on C; and (ii) for every $P_i \in D$, $j \in Ag$, and $\vec{u} \in U^{q_i}$, $\vec{u} \in l_j(P_i)$ iff $\iota(\vec{u}) \in l'_j(P_i)$.

Any function ι as above is a *witness* for $s \simeq s'$. Notice that isomorphic instances preserve first-order (non-modal) formulas:

Proposition 1. Let φ be an FO-formula, assume that $s \simeq s'$, and let $\sigma : Var \mapsto U$ and $\sigma' : Var \mapsto U'$ be assignments s.t. (i) there is a bijection $\gamma : ad(s) \cup C \cup \sigma(fr(\varphi)) \mapsto ad(s') \cup C \cup \sigma'(fr(\varphi));$ (ii) γ is a witness for $s \simeq s'$; and (iii) $\sigma' = \gamma \circ \sigma$. Then $(\mathcal{P}, s, \sigma) \models \varphi$ iff $(\mathcal{P}', s', \sigma') \models \varphi$.

Essentially, Prop. 1 says that isomorphic instances cannot distinguish FO-formulas. We now generalise this result to the language FO-CTLK.

Definition 15 (Similarity). The AC-MAS \mathcal{P}' simulates \mathcal{P} , or $\mathcal{P} \preceq \mathcal{P}'$, iff there exists a simulation relation on $\mathcal{S} \times \mathcal{S}'$, i.e., a relation $\preceq s.t.$ (i) $s_0 \preceq s'_0$; and (ii) if $s \preceq s'$ then

- 1. $s \simeq s'$;
- 2. for every t, if $s \to t$ then there is t' s.t. $s' \to t'$, $s \oplus t \simeq s' \oplus t'$ and $t \preceq t'$;
- 3. for every t, if $s \sim_i t$ then there is t' s.t. $s' \sim_i t'$, $s \oplus t \simeq s' \oplus t'$ and $t \preceq t'$.

Moreover, we say that \mathcal{P} and \mathcal{P}' are *bisimilar*, or $\mathcal{P} \approx \mathcal{P}'$, iff there exists a *bisimulation relation* on $\mathcal{S} \times \mathcal{S}'$, i.e., a relation \approx s.t. both \approx and $\approx^{-1} = \{\langle s', s \rangle \mid s \approx s'\}$ are simulation relations.

Finally, we introduce the class of AC-MAS of interest.

Definition 16 (Uniformity). An AC-MAS \mathcal{P} is uniform iff for every $s, t, s' \in \mathcal{S}$, $t' \in \mathcal{D}(U)$, if $t \in \tau(s, \alpha(\vec{u}))$ and $s \oplus t \simeq s' \oplus t'$ for some witness ι , then for every bijection ι' extending $\iota, t' \in \tau(s', \alpha(\iota'(\vec{u})))$.

Intuitively, uniformity requires that transitions do not depend on the data content of each state, apart from constants in C. This definition of uniformity extends [4] as parametric actions are considered explicitly, thus allowing for effective abstraction.

We now show that uniformity, together with bisimilarity and boundedness, is sufficient to preserve FO-CTLK formulas, where an AC-MAS \mathcal{P} is *b*-bounded, for $b \in \mathbb{N}$, if for all $s \in S$, $|ad(s)| \leq b$. Observe that boundedness imposes no restriction on the domain U of \mathcal{P} . Thus, if U is infinite, so is the state space of \mathcal{P} in general.

The next results show that, although infinite-state, an uniform and b-bounded AC-MAS \mathcal{P} can be verified by model checking the *finite abstraction* of \mathcal{P} . In what follows $N_{Ag} = \max_{\alpha(\vec{x}) \in Act} \{ |\vec{x}| \}.$

Definition 17. Let Ag be a set of agents, and let U' be a set. For each agent i = $\langle \mathcal{D}, L, Act, Pr \rangle$ in Ag we define an agent $i' = \langle \mathcal{D}', L', Act', Pr' \rangle$ s.t. (i) $\mathcal{D}' = \mathcal{D}$; (ii) $L' = \mathcal{D}'(U')$; (iii) Act = Act; (iv) $\alpha(\vec{u}) \in Pr'(l')$ iff there is $l \in L$ s.t. $l' \simeq l$ for some witness ι , and $\alpha(\iota'(\vec{u})) \in Pr(l)$ for some bijection ι' extending ι . Let Ag' be the set of all i' thus defined.

Notice that the definition of i' depends on the set U'. However, we omit U' when it is clear from the context.

Definition 18 (Abstraction). Let \mathcal{P} be an AC-MAS over Ag, the abstraction \mathcal{P}' over *Ag'* is defined as follows:

- $s'_0 = s_0$; $t' \in \tau'(s', \alpha(\vec{u}))$ iff there are s, t, and \vec{u}' s.t. $t \in \tau(s, \alpha(\vec{u}'))$, $s \oplus t \simeq s' \oplus t'$ for some witness ι , and $\vec{u} = \iota'(\vec{u}')$ for some bijection ι' extending ι ;
- S' is the set of reachable states.

Notice that \mathcal{P}' is an AC-MAS. In particular, \mathcal{P}' satisfies the conditions on protocols and transitions, and it is finite whenever U' is.

We can now prove the main result of this section, which extends Theorem 4.7 in [4] to AC-MAS with parametric actions.

Theorem 1. Given an infinite, b-bounded and uniform AC-MAS \mathcal{P} , an FO-CTLK formula φ , and a finite set $U' \supseteq C$ s.t. $|U'| \ge 2b + |C| + \max\{var(\varphi), N_{Aq}\}$, the abstraction \mathcal{P}' is finite, uniform and bisimilar to \mathcal{P} . In particular, $\mathcal{P} \models \varphi$ iff $\mathcal{P}' \models \varphi$

This result states that by using a sufficient number of elements in \mathcal{P}' , we can reduce the verification of an infinite AC-MAS to verifying a finite one. Also notice that U'can be taken to be any finite subset of U satisfying the condition on cardinality. By doing so, the finite abstraction \mathcal{P}' can be defined simply as the restriction of \mathcal{P} to U'. Thus, every infinite, b-bounded and uniform AC-MAS is bisimilar to a proper finite subsystem, which can be effectively generated.

5 Embedding of GSM Programs into AC-MAS

In this section we introduce an embedding of GSM programs into AC-MAS. By doing so we achieve two results. Firstly, we provide a formal semantics to GSM programs via AC-MAS that can be used to interpret FO-CTLK specifications. Secondly, this enables us to apply the finite abstraction methodology in Section 4 to GSM programs.

To begin with, for each artifact type $AT = \langle P, x, Att, Stg, Mst, Lcyc \rangle$ we introduce a predicate symbol P with attributes x, Att. Hence, the arity of P is $q_P =$ 1 + |Att|.

Definition 19. Given a GSM program $\Gamma = \{AT_j\}_{j \leq n}$ we define a database schema $\mathcal{D}_{\Gamma} = \{P_j\}_{j \leq n}$ s.t. each P_j is the predicate symbol corresponding to the artifact type AT_i .

We now introduce agents in GSM programs.

Definition 20. Given a GSM program Γ and an interpretation domain U, an agent is a tuple $i = \langle \mathcal{D}_i, L_i, Act_i, Pr_i \rangle$ s.t.

- $\mathcal{D}_i \subseteq \mathcal{D}_{\Gamma}$ is the local database schema, and $\mathcal{D}_E = \mathcal{D}_{\Gamma}$;
- $L_i = \mathcal{D}_i(U)$ is the set of local states, and $L_E = \mathcal{D}_{\Gamma}(U)$;
- Act_i is the set of actions $\alpha_e(\vec{y})$ for each event type e with formal payload \vec{y} . Further, we introduce a skip action $skip_i$ for each agent i. Act_E is defined similarly.
- For every ground action $\alpha_i(\vec{u})$, for every local state l_i , $\alpha_i(\vec{u}) \in Pr_i(l_i)$, i.e., a ground action $\alpha_i(\vec{u})$ is always enabled.

We observe that the original formulation of GSM programs in [10] does not account for agents. In fact, artifacts are bundled together in the *Artifact Service Center* (ASC), which interacts with the external environment through incoming and generated events. According to Def. 20 the Artifact Service Center of GSM programs is mapped into the environment of AC-MAS; while the environment of GSM programs is mapped to agents in AC-MAS. So, the notion of environment corresponds to different entities in GSM programs and AC-MAS. We keep the original terminology, as the distinction is clear. Furthermore, each agent, including the environment, perform actions corresponding to sending an event to the ASC. As illustrated in Section 2.1, these include 1-way messages M, 2-way service call returns F^{return} , and artifact instance creation requests create^{call} Actions are always enabled as no protocol is explicitly given for GSM programs.

Given a set of agents defined as above, the AC-MAS \mathcal{P}_{Γ} associated to the GSM program Γ is defined according to Def. 11. Specifically,

Definition 21. Given a set Ag of agents over the GSM program Γ and a snapshot Σ_0 , the AC-MAS associated with Γ is a tuple $\mathcal{P}_{\Gamma} = \langle S, U, s_{\Sigma_0}, \tau \rangle$ s.t.

- $S \subseteq L_e \times L_1 \times \cdots \times L_n$ is the set of reachable global states;
- *U* is the interpretation domain;
- $s_{\Sigma_0} \in S$ is the initial global state corresponding to Σ_0 ;
- $\tau : S \times Act(U) \mapsto 2^S$ is the global transition function s.t. $t \in \tau(s, \alpha(\vec{u}))$ iff (i) if $\alpha = \langle \alpha_e, \alpha_1, \dots, \alpha_n \rangle$ then at most one α_i is different from $skip_i$; (ii) if $\alpha_i = \alpha_e$ then (Σ_s, e, Σ_t) holds in Γ , where \vec{u} is the payload of event e.

Notice that, given a set Ag of agents, there is a one-to-one correspondence between snapshots in Γ and states in the AC-MAS \mathcal{P}_{Γ} . Given a snapshot Σ we denote the corresponding state as s_{Σ} . Similarly, Σ_s is the snapshot corresponding to the global state s. Also, GSM programs do not specify initial states; therefore the definition of \mathcal{P}_{Γ} is parametric in Σ_0 , i.e., the snapshot chosen as the initial state of Γ . Most importantly, the transition function τ mirrors the B-step semantics of GSM programs. Since each B-step consumes a single event, we require that at most one agent performs an event action at each given time, while all other agents keep idle. This has correspondences with other approaches based on multi-agent systems, such as interleaved interpreted systems [16].

5.1 Finite abstractions of GSM programs

In this section we show that GSM programs admit finite abstractions. Specifically, by suitably restricting the language of sentries we can prove that the AC-MAS \mathcal{P}_{Γ} obtained from a GSM program Γ is uniform. So, by applying Theorem 1 we obtain that if \mathcal{P}_{Γ} is also bounded, then it admits a finite abstraction, hence its model checking problem is decidable. The key notion of this subsection is that of *amenable* GSM program. Hereafter, $\mathcal{L}_{\mathcal{D}_{\Gamma}}$ is the first-order (non-modal) language of formulas built on the predicate symbols in the database schema \mathcal{D}_{Γ} in Def. 19.

Definition 22 (Amenable GSM programs). A sentry $\chi(x)$ is amenable iff the FOformula $\varphi(x)$ in $\chi(x)$ belongs to the language $\mathcal{L}_{D_{\Gamma}}$. A GSM program is amenable iff all sentries occurring in any guard or milestone are amenable.

It is known that, given a database schema D, the language \mathcal{L}_D built on it is sufficiently expressive to define a wide range of systems [4, 14]. As an example, the scenario in Section 3 adheres to this property. Therefore we see amenable GSM programs as an interesting and powerful class of GSM programs with potentially wide applicability.

The next results show that the AC-MAS \mathcal{P}_{Γ} is uniform whenever Γ is amenable.

Lemma 1. For every states $s, t \in \mathcal{P}_{\Gamma}$, if $s \simeq t$ for some witness ι , then $\Sigma_t = \iota(\Sigma_s)$.

Proof. Notice that if ι is a witness for $s \simeq t$, then in particular the attributes x and Att in Σ_s are mapped to the corresponding attributes in Σ_t . Further, the attributes in Stg, Mst and Lcyc remain the same.

The next result is key in the proof of uniformity for \mathcal{P}_{Γ} .

Lemma 2. For every $s, t, s' \in S, t' \in D_{\Gamma}(U)$, if $s \oplus t \simeq s' \oplus t'$ for some witness ι , then (Σ_s, e, Σ_t) implies $(\Sigma_{s'}, \iota'(e), \Sigma_{t'})$ where ι' is any bijection extending ι

Proof. Assume that (Σ_s, e, Σ_t) and ι is a witness for $s \oplus t \simeq s' \oplus t'$. We show that $(\Sigma_{s'}, \iota'(e), \Sigma_{t'})$ where ι' is a bijection extending ι . If (Σ_s, e, Σ_t) then there is a sequence $\Sigma_0, \ldots, \Sigma_k$ of snapshots s.t. $\Sigma_0 = \Sigma_s, \Sigma_1 = ImmEffect(\Sigma_s, e)$, and $\Sigma_k = \Sigma_t$. Also, for $1 \leq j \leq k - 1, \Sigma_{j+1}$ is obtained from Σ_j by the application of a PAC rule. We show that we can define a sequence $\Sigma'_0, \ldots, \Sigma'_k$ s.t. $\Sigma'_0 = \Sigma_{s'},$ $\Sigma'_1 = ImmEffect(\Sigma_{s'}, \iota'(e)), \Sigma'_k = \Sigma_{t'},$ and for $1 \leq j \leq k - 1, \Sigma'_{j+1}$ is obtained from Σ'_j by the application of a PAC rule. This is sufficient to show that $(\Sigma_{s'}, \iota'(e), \Sigma_{t'})$. First, for $0 \leq j \leq k$ define $\Sigma'_j = \iota'(\Sigma_j)$. By Lemma 1 we have that $\Sigma'_0 = \iota'(\Sigma_s) = \Sigma_{s'}$ and $\Sigma'_k = \iota'(\Sigma_t) = \Sigma_{t'}$. Also, it is clear that if $\Sigma_1 = ImmEffect(\Sigma_s, e)$, then we have that $\Sigma'_1 = \iota'(\Sigma_1) = \iota'(ImmEffect(\Sigma_s, e))$ is equal to $ImmEffect(\iota'(\Sigma_s), \iota'(e)) =$ $ImmEffect(\Sigma_{s'}, \iota'(e))$. Finally, we show that if Σ_{j+1} is obtained from Σ_j by an application of a PAC rule, then also Σ'_{j+1} is obtained from Σ'_j by the same PAC rule. Consider the PAC rule $\langle \pi(x), \alpha(x), \gamma(x) \rangle$. We have that if $\Sigma_s \models \pi(\rho)$ for some artifact ID ρ in Σ_s , then clearly $\Sigma_{s'} \models \pi(\iota'(\rho))$. Further, let $\Sigma_j \models \alpha(\rho) \equiv \chi(\rho) \land \psi(\rho)$, where $\chi(x)$ is an amenable sentry and $\psi(x)$ is of the form $\tau.m, \neg \tau.m, \tau.active_s$, or $\neg \tau.active_s$. Clearly, if $\Sigma_j \models \psi(\rho)$ then $\Sigma'_j \models \psi(\iota'(\rho))$. Further, since $\chi(x)$ is of the form on $\xi(x)$ if $\varphi(x)$ and $\varphi(x)$ is an FO-formula in $\mathcal{L}_{D_{\Gamma}}$, then by Prop. 1 we have that $\Sigma'_j \models \chi(\iota'(\rho))$. Hence, $\Sigma'_{j} \models \alpha(\iota'(\rho))$. Finally, if Σ_{j+1} is constructed from Σ_{j} by applying the effect called for by $\gamma(\rho)$, then Σ'_{j+1} is constructed from Σ'_{j} by applying the effect called for by $\gamma(\iota'(\rho))$. Thus, we have the desired result.

Lemma 2 enables us to state the first of our two key results.

Theorem 2. If the GSM program Γ is amenable, then the AC-MAS \mathcal{P}_{Γ} is uniform.

Proof. Let assume that $t \in \tau(s, \alpha(\vec{u}))$ for some ground action $\alpha(\vec{u}) \in Act(U)$, and $s \oplus t \simeq s' \oplus t'$ for some witness ι . We prove that $t' \in \tau(s', \alpha(\iota'(\vec{u})))$, where ι' is a bijection extending ι . By the definition of τ in \mathcal{P}_{Γ} , $t \in \tau(s, \alpha(\vec{u}))$ if (Σ_s, e, Σ_t) , where e is a ground event with payload \vec{u} , and $\alpha_i = \alpha_e$ for exactly one of the components in α . By Lemma 2 we have that $(\Sigma_{s'}, \iota'(e), \Sigma_{t'})$, and again by definition of τ we obtain that $t' \in \tau(s', \alpha(\iota'(\vec{u})))$. As a result, \mathcal{P}_{Γ} is uniform.

By combining Theorems 1 and 2 we obtain a decidable model checking procedure for amenable GSM programs. Specifically, a GSM program Γ is *b*-bounded if the cardinality of all snapshots is bounded, i.e., there is a $b \in \mathbb{N}$ s.t. $|\Sigma| \leq b$ for all snapshots $\Sigma \in \Gamma$. Hence, we have the following result:

Corollary 1. Assume a b-bounded and amenable GSM program Γ on an infinite domain U, an FO-CTLK formula φ , and a finite set $U' \supseteq C$ s.t. $|U'| \ge 2b + |C| + \max\{var(\varphi), N_{Ag}\}$. Then, the abstraction \mathcal{P}' of \mathcal{P}_{Γ} is uniform and bisimilar to \mathcal{P}_{Γ} . In particular, $\mathcal{P}_{\Gamma} \models \varphi$ iff $\mathcal{P}' \models \varphi$.

Thus, to verify a GSM program we can model check the finite abstraction of the corresponding AC-MAS. Notice that by the remarks at the end of Section 4.1 the latter procedure can be computed effectively.

To conclude, in [4] it was proved that the model checking problem for finite AC-MAS is PSPACE-complete in the size of the state space S and the specification φ . So, we obtain the following:

Proposition 2. *Model checking bounded and amenable GSM programs is in PSPACE in the number of states in the finite abstraction and the length of the specification.*

Notice that amenability is a sufficient condition for decidability, but we have no hint it is also necessary. So, the property of having a finite abstraction could be enjoyed by a larger class of GSM programs. This point demands further investigations.

5.2 The RPO Scenario as an AC-MAS

We now show how the GSM program RPO for the Requisition and Procurement Orders scenario in Section 3 translates into the corresponding AC-MAS \mathcal{P}_{RPO} . Firstly, we associate the RPO program with the database schema \mathcal{D}_{RPO} containing a predicate symbol P_{RO} for the Requisition Order artifact type, as well as a predicate symbol P_{PO} for the Procurement Order artifact type. In particular, the predicate symbol P_{RO} has data and status attributes as specified in the datamodel in Fig. 1. The definition of P_{PO} is similar. A number of agents appears in the RPO scenario: a Customer C, a Manufacturer M, and one or more Suppliers S. According to Def. 20 each agent has a partial view of the database schema $\mathcal{D}_{RPO} = \{P_{RO}, P_{PO}\}$. We can assume that the Customer can only access the Requisition Order (i.e., $\mathcal{D}_{C} = \{P_{RO}\}$), and the Supplier only the Procurement Order (i.e., $\mathcal{D}_{S} = \{P_{PO}\}$), while the Manufacturer can access both (i.e., $\mathcal{D}_{M} = \{P_{RO}, P_{PO}\} = \mathcal{D}_{RPO}$). Finally, the AC-MAS $\mathcal{P}_{RPO} = \langle S, U, s_0, \tau \rangle$ defined according to Def. 21, is designed to mimic the behaviour of the *RPO* program. In particular, S is the set of reachable states; U is the interpretation domain containing relevant items and data; s_0 is an initial state, for instance, the one where all relations are empty; τ is the transition function as in Def. 21. We define a temporal transition $s \to s'$ in \mathcal{P}_{RPO} iff there is some ground event e s.t. $\langle \Sigma_s, e, \Sigma_{s'} \rangle$ holds in *RPO*.

By means of the AC-MAS \mathcal{P}_{RPO} we can model check the RPO program against first-order temporal epistemic specifications. For instance, the following FO-CTLK formula specifies that the manufacturer M knows that each Procurement Order has to match a corresponding Requisition Order:

$$\phi = AG \; \forall ro_id, \vec{x} \; (PO(id, ro_id, \vec{x}) \to K_{\mathsf{M}} \; \exists \vec{y} \; RO(ro_id, \vec{y}))$$

We remark without a formal proof that the RPO program can be defined so that any clause $\varphi(x)$ in any sentry $\chi(x)$ belongs to the FO-language $\mathcal{L}_{\mathcal{D}_{RPO}}$. Actually, first-order languages on databases have proved significantly expressive in a number of context [4, 14]. Hence, the RPO program is amenable, and by Theorem 2 the AC-MAS \mathcal{P}_{RPO} is uniform. Finally, if we also assume that the RPO program is bounded, then according to Def. 18 we can introduce the finite abstraction \mathcal{P}' of \mathcal{P}_{RPO} . This can be effectively constructed as the subsystem \mathcal{P}' of \mathcal{P}_{RPO} defined on a finite subset of the interpretation domain satisfying the cardinality condition, that is, \mathcal{P}' is defined as \mathcal{P}_{RPO} but for the domain of interpretation U', which can be taken as the finite $U' \supseteq C$ s.t. |U| = 2b + $|C| + \max\{var(\phi), N_{Ag}\}$. By Corollary 1 we can check whether the RPO program satisfies ϕ by model checking the finite abstraction \mathcal{P}' .

This leaves open the problem of checking that the RPO program is actually bounded, a problem of general relevance to verification (think, e.g., of checking whether a program may run out of memory), yet orthogonal to ours. A partial answer to this is provided by [14], which isolates a sufficient condition that guarantees boundedness of processes operating on artifacts.

6 Conclusions

This paper stems from an observation regarding the current lack of support for verification in GSM environments. While abstraction methodologies for various artifactinspired systems have been put forward [14, 4, 8], they all lack support for program verification and operate on logical models, thereby making automatic model checking impracticable. Our objective in this paper was to surpass this severe limitation and position the GSM approach firmly in an agent-based semantics, so that information-theoretic properties such as knowledge of the participants could be verified. We achieved this by extending minimally the semantics of AC-MAS [4] to account for parametric actions, while at the same time maintaining the key results concerning finite abstractions. We then proceeded to map constructs of GSM into the corresponding constructs of the revised AC-MAS, and identified what we called "amenable" GSM programs that we showed to admit finite abstractions. We remarked that amenability is not a significant limitation in applications and showed the key passages of the approach presented on a fraction of a notable use-case from [10]. The work in this paper brings us considerably closer to an implementation for automatic model checking of GSM programs [13], which is what we intend to pursue next.

Acknowledgements. This research was supported by the EC through the STREP Project "ACSI" (grant no. 257593), and by the UK EPSRC Leadership Fellowship "Trusted Autonomous Systems" (grant no. EP/I00520X/1).

References

- 1. S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. Addison-Wesley, 1995.
- 2. F. Belardinelli, A. Lomuscio, and F. Patrizi. A Computationally-Grounded Semantics for Artifact-Centric Systems and Abstraction Results. In *Proc. of IJCAI*, 2011.
- 3. F. Belardinelli, A. Lomuscio, and F. Patrizi. Verification of Deployed Artifact Systems via Data Abstraction. In *Proc. of ICSOC*, 2011.
- 4. F. Belardinelli, A. Lomuscio, and F. Patrizi. An Abstraction Technique for the Verification of Artifact-Centric Systems. In *Proc. of KR*, 2012.
- D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. View-based query answering in description logics: Semantics and complexity. J. Comput. Syst. Sci., 78(1):26–46, 2012.
- D. Cohn and R. Hull. Business Artifacts: A Data-Centric Approach to Modeling Business Operations and Processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.
- E. Damaggio, R. Hull, and R. Vaculín. On the Equivalence of Incremental and Fixpoint Semantics for Business Artifacts with Guard-Stage-Milestone Lifecycles. In *Proc. of BPM*, 2011.
- A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic Verification of Data-Centric Business Processes. In *Proc. of ICDT*, 2009.
- Alin Deutsch, Liying Sui, and Victor Vianu. Specification and Verification of Data-Driven Web Applications. J. Comput. Syst. Sci., 73(3):442–474, 2007.
- R. Hull et al. Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events. In *Proc. of DEBS*, 2011.
- R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning About Knowledge*. The MIT Press, 1995.
- 12. P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proc. of CAV*, 2004.
- 13. P. Gonzalez, A. Griesmayer, and A. Lomuscio. Verifying gsm-based business artifacts. In *Proceedings of the 19th International Conference on Web Services (ICWS12), Honolulu, USA*. IEEE Press, 2012.
- 14. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, R. De Masellis, and P. Felli. Foundations of Relational Artifacts Verification. In *Proc. of BPM*, 2011.
- M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, B. Wozna, and A. Zbrzezny. Verics 2007 - a model checker for knowledge and real-time. *Fundamenta Informaticae*, 85(1-4):313–328, 2008.
- A. Lomuscio, W. Penczek, and H. Qu. Partial Order Reductions for Model Checking Temporal-epistemic Logics over Interleaved Multi-agent Systems. *Fundamenta Informaticae*, 101(1-2):71–90, 2010.
- A. Lomuscio, H. Qu, and F. Raimondi. Mcmas: A model checker for the verification of multi-agent systems. In *Proc. of CAV*, 2009.