Verification of GSM-based Artifact-centric Systems by Predicate Abstraction

Pavel Gonzalez[1], Andreas Griesmayer[2], Alessio Lomuscio[1]

[1] Department of Computing, Imperial College London
{pavel.gonzalez09, a.lomuscio}@imperial.ac.uk
[2] ARM, Cambridge
andreas.griesmayer@arm.com

Abstract. Artifact-centric systems are a recent paradigm to model and implement business workflows. They describe data, processes, internal and external agents and include mechanisms for data hiding and access control. GSM is a language for the implementation of artifact-centric systems. Since GSM programs have infinitely many states, their verification is challenging. We here present a predicate abstraction technique that enables us to verify GSM programs against rich specifications built on an epistemic, first-order variant of the μ-calculus. We give the theoretical underpinnings of the technique and present GSMC, the first model checker for GSM that implements SMT-based, three-valued abstraction for GSM.

1 Introduction

Artifacts are structures that "combine data and process in an holistic manner" to describe business interactions, typically in a service-oriented architecture [1]. The data component is given by the relational databases underpinning the artifacts in a system, whereas the workflows are described by "lifecycles" associated with each artifact schema. Artifacts systems define complex workflow schemes based on artifacts. The system's participants, or agents, interact with the artifact system by performing events on it.

Differently from services where typically only the process interfaces are advertised, in artifact-centric systems the data structures are also made public. Due to their expressiveness and flexibility, Artifact-centric architectures are increasingly being used in variety of application areas including case management systems [2]. Artifact centric systems are executed in a hub which provides the functionality for service execution. A flexible and powerful language for modelling and executing artifact-centric systems is the Guard-Stage-Milestone programming language (GSM). The open-source design and runtime engine `Acsi Hub` [3, 4] is an environment whereby system orchestration and choreography are executed.

If artifact-centric environments are to fulfil their promise to drive the future generation of data-intensive services, they need to be verifiable. This should involve not only the hub itself governing the interactions between artifact calls, but also, and crucially, the agents implementing the services in the system, as is normally done when reasoning about services [5]. In addition to providing correctness guarantees and rapid prototyping, techniques such as model checking can form the underpinnings for the implementation of automatic service orchestration and choreography [6].

In this paper we develop verification methodologies for artifact-centric systems implemented in GSM. Since GSM programs include data models, they are infinite state programs; it follows that traditional model checking methods based on finite-state machines cannot be applied to them. To address this problem we develop a novel predicate abstraction methodology [7] for GSM defined on a three-valued semantics to account for over- and under-approximation of the models. We also present GSMC, the first model checker for GSM, that implements the technique discussed. We evaluate the technique on a large industrial scale example.

Related Work. Several techniques for the verification of artifact-centric systems have been put forward [8–13]. While these provide considerable insight in the decidability and complexity of the verification problem, they do not provide a concrete verification technique for actual systems. The first contributions concerning the practical verification of GSM systems appeared in [14, 15]. These, however, are defined on coarse, user-given abstractions of GSM models where little data is present and ad-hoc restrictions on variable ranges are applied to obtain finite state systems. Additionally the specification language used is limited.

Incomplete verification methodologies operating directly on the source code have been developed in software verification. The abstraction techniques developed in this context normally target reachability properties only. However, 3-valued abstraction can be applied to specifications based on the μ-calculus [16].

This paper extends existing work by providing 3-valued abstractions for GSM programs specified by a first-order version of the epistemic μ calculus. This enables us to specify services not in purely propositional terms as it is traditionally done but, instead, by referring to the underlying databases.

2 The Guard-Stage-Milestone language and Multi-Agent Systems

While GSM provides a language for the realisation of artifact-centric systems, GSM on its own is not equipped with constructs for the implementation of external actors operating on the system. In GSM these are abstracted by events reaching the system.

However, to verify the possible executions of the system we need to represent how the agents interact with it. Artifact-centric Multi-Agent Systems (AC-MAS) were put forward in [15] to provide a semantics for GSM and the behaviours of external agents. We summarise these concepts below but refer to the cited literature for more details.

The Guard-Stage-Milestone (GSM) has recently been put forward as a declarative language for implementing artifact systems [17]. GSM describes an artifact system Γ that depends on of *artifact types* that correspond to classes of key business entities. A system comprises of a number of *artifact instances* of artifact types. Each type has an *information model*, which gives an integrated view of the business data, and a hierarchical *lifecycle model*, which describes the structure and evolution of the business process. The artifact system interacts with its environment via *events*. The *information model* is partitioned into the set of *data attributes*, which hold business data, and the set of *status attributes*, which capture the state of the lifecycle model. Figure 1 illustrates a portion of the lifecycle of a manufacturing process and represents the core concepts: The boxes denote *stages*, which represent clusters of activity designed to achieve milestones ($\circ$)

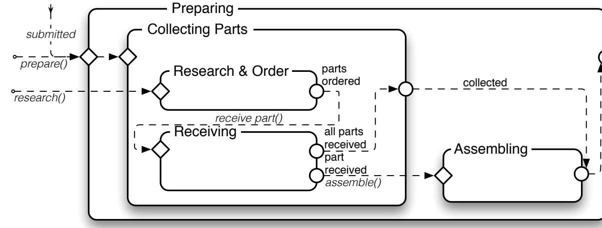

Fig. 1. A lifecycle model.

that represent operational objectives. A *guard* ($\diamond$) triggers activities in a stage when a certain condition is fulfilled. Both milestones and guards are controlled declaratively through *sentries*. A sentry of an artifact instance ι is an expression $\chi(\iota)$ in terms of incoming events, guards and milestones, and the status of the instance. In the example above, the Stage ‘Collecting Parts’ contains ‘Research & Order’, which is triggered by an external event; upon reaching the milestone ‘parts ordered’ the next stage ‘Receiving’ is activated.

The operational semantics for GSM is based on the notion of a *business step* (B-step). This is an atomic unit that corresponds to the effect of processing one incoming event. A B-step has the form of a tuple $\sigma = (\Sigma, e, \Sigma')$, where e is an incoming external event and Σ, Σ' are snapshots that capture the current and next state of the information model respectively.

The programming language GSM [4] provides the construct for the realisation of GSM systems; the semantics of GSM programs is given in terms of B-steps.

Artifact-Centric Multi-Agent Systems. While GSM models the business artifacts, agents model the possible interactions that external actors and services may have with the artifact system. Below we summarise the key elements from [15] where a formalism for defining the behaviour of the agents, and their access to the artifact system, is described. The concepts of *views* and *windows* are used define which attributes and artifact instances are visible to an agent; *events* represent external actions that cause a change in the system. In the example above, *views* can be used to hide details like procurement of parts from a customer, while allowing access to higher level information, e.g., the start and end of the parts assembling process. textitWindow, instead, can be used to hide orders that do not belong to a particular customer. While a view ν and an event ϵ are simple lists, a window $\omega_i(\iota)$ is a formula that is evaluated for a specific artifact instance ι and an agent i. The instance is exposed to the agent only if $\omega_i(\iota)$ evaluates to *true*. The behaviour of an agent is given by its *protocol* $\wp$ in terms of the visible state of the artifact system, and the agent's unique ID and set of private variables `var`.

We formalise an *agent-based GSM system* for a set of agents $\mathcal{A}$ operating on an environment given by the artifact system E through an Artifact-centric Multi-Agent Systems (AC-MAS)[9]. An AC-MAS $\mathcal{P} = \langle S, \mathcal{I}, Act, \tau, \Lambda \rangle$, where $S \subseteq L_E \times L_1 \times \cdots \times L_n$ is the set of *reachable global states*, $\mathcal{I}$ is the *initial state*, $Act = Act_E \times Act_1 \times \cdots \times Act_n$ is the set of actions, $\tau : S \times Act \to 2^S$ is the *global transition relation*, and $\Lambda : S \to 2^{AP}$ is the *evaluation relation* for a set of propositions AP. A global state $(l_E, l_1, \ldots, l_n) \in S$ for the system is given in terms of the snapshot Σ of

the artifact system for l_E, and the accessible variables of each agent for $l_1, \ldots, l_n$. We also write $l_i(s)$ to extract the visible state for agent i from a global state $s \in S$. The sets of actions Act_E and Act_i are directly defined by the events the system provides and the permissions of the agents. The global transition relation $\tau(s, \alpha)$ with $s \in S$ and $\alpha \in Act$ is given by the corresponding B-steps defined by GSM in combination with the protocols $\wp$ of the agents, where only one agent can interact with the artifact system at a time while the others are idle.

The initial state $\mathcal{I}$ is a global state with not artifact instances in Σ and with all private variables set to their initial value. We write $s \rightarrow s'$ iff there exists an α, such that $s' \in \tau(s, \alpha)$; in this case s' a *successor* of s. A *run* r from s is an infinite sequence $s^0 \rightarrow s^1 \rightarrow \ldots$ with $s^0 = s$. We write $r[i]$ for the i-th state in the run and r_s for the set of all runs starting from s. A state s' is *reachable* from s if there is a run from s that contains s', formally $\exists r' \in r_s : \exists i \geq 0 : r'[i] = s'$. Note that portions of the global state may not be visible to an agent. In line with the standard semantics of epistemic logic [18], we say that the states s and s' are *epistemically indistinguishable* for agent i, or $s \sim_i s'$, iff $l_i(s) = l_i(s')$, i.e., if agent i's local state is the same in s and s'.

3 Three-Valued Abstraction for AC-MAS

Predicate abstraction [7] is a technique used to generate sound approximations of infinite state systems by grouping together system states satisfying certain properties into *abstract states*. *May* transition between abstract states correspond to possible transitions between some of corresponding concrete states. This leads to an over-approximation of the possible behaviour that is *conservative* for safety properties but may lead to unsound results otherwise. Three-valued abstraction has been employed [19, 16] to overcome these limitations. In three-valued abstraction a second transition relation (or *must* relation) is introduced to encode when a change in the corresponding concrete states must happen. This allows to concurrently maintain over- and under-approximations that are conservative for both positive and negative specifications and allows to detect when a result cannot be determined.

To extend this technique to AC-MAS, we introduce the three-valued semantics for the epistemic μ-calculus and replace τ with τ_m, the *global may transition relation*, and τ_M, the *global must transition relation*, to get $\mathcal{P} = \langle S, \mathcal{I}, Act, \tau_m, \tau_M, \Lambda \rangle$. Analogously to the concrete case, we write $s \xrightarrow{\text{may}} t$ ($s \xrightarrow{\text{must}} t$) for $t \in \tau_m(s, a)$ ($t \in \tau_M(s, a)$). Over- and under-approximations for the epistemic relations are denoted as $\overset{\text{may}}{\sim}_i$ and $\overset{\text{must}}{\sim}_i$ respectively. This extended definition of AC-MAS allows us to define abstraction formally as:

Definition 1 (Abstraction). *Let $\mathcal{P} = \langle S, \mathcal{I}, Act, \tau_m, \tau_M, \Lambda \rangle$ and $\mathcal{P}' = \langle S', \mathcal{I}', Act', \tau'_m, \tau'_M, \Lambda' \rangle$ be AC-MAS over the same set $\mathcal{A}$ of agents and sets $AP' \subseteq AP$ of propositions. We say that $\mathcal{P}'$ is an* abstraction *of $\mathcal{P}$ if:*

1. *$s' \in \mathcal{I}'$ iff there exists $s \in \mathcal{I}$, such that $s \in \gamma(s')$;*
2. *$s' \xrightarrow{\text{may}}{}' t'$ iff there exist $s \in \gamma(s')$ and $t \in \gamma(t')$, such that $s \xrightarrow{\text{may}} t$;*
3. *$s' \xrightarrow{\text{must}}{}' t'$ iff for each $s \in \gamma(s')$ there exists $t \in \gamma(t')$, such that $s \xrightarrow{\text{must}} t$;*
4. *$s' \overset{\text{may}}{\sim}{}'_i t'$ iff there exist $s \in \gamma(s'), t \in \gamma(t')$ such that $s \overset{\text{may}}{\sim}_i t$ or there exists u' such that $s' \overset{\text{may}}{\sim}_i u'$ and $u' \overset{\text{may}}{\sim}_i t'$;*

5. $s' \overset{must}{\sim}_i{}' t'$ *iff for each* $s \in \gamma(s')$ *there exists* $t \in \gamma(t')$*, such that* $s \overset{must}{\sim}_i t$*, and for each* $t \in \gamma(t')$ *there exists* $s \in \gamma(s')$*, such that* $t \overset{must}{\sim}_i s$*;*
6. $p \in \Lambda'(s')$ *iff* $p \in \Lambda(s)$ *for each* $s \in \gamma(s')$*;*

where $\gamma : S' \mapsto 2^S$ *is the* concretisation function *that maps each abstract state* $s' \in S'$ *to the non-empty set of concrete states* $S_{s'} \subseteq S$ *it represents;* $\overset{may}{\longrightarrow}{}'$ *and* $\overset{may}{\longrightarrow}$ *are the* may *transition relations in* $\mathcal{P}'$ *and* $\mathcal{P}$ *respectively;* $\overset{must}{\longrightarrow}{}'$ *and* $\overset{must}{\longrightarrow}$ *are the* must *transition relations;* $\overset{may}{\sim}_i{}'$ *and* $\overset{may}{\sim}_i$ *are the* may *epistemic relations; and* $\overset{must}{\sim}_i{}'$ *and* $\overset{must}{\sim}_i$ *are the* must *epistemic relations.*

May transition relations in the abstract model $\mathcal{P}'$ over-approximate *may* transition relations in the concrete model $\mathcal{P}$: whenever there is a *may* transition between two states in $\mathcal{P}$, there is is a transition between the corresponding abstract states of $\mathcal{P}'$. Conversely, *must* transition relations in the abstract model $\mathcal{P}'$ under-approximate *must* transition relations in the concrete model $\mathcal{P}$; they are only created for concrete transitions that are common to all of the states of $\mathcal{P}$ represented by the source abstract state.

We define may and must epistemic possibility relations in the abstract system similarly to the temporal case; however, there are additional constraints due to the nature of the relations. Specifically, we require both to be equivalence relations. This is achieved by building the transitive closure for $\overset{may}{\sim}_i$, while relations in $\overset{must}{\sim}_i$ that are not symmetric are removed. By insisting on equivalence relations, we ensure that the usual KT45 axioms [18] for knowledge are satisfied in the abstract model.

Note that if the abstract *may* epistemic possibility relation were defined analogously to abstract *may* transition relations, it would not necessarily be transitive. Therefore, we define the abstract *may* epistemic possibility relation as the transitive closure of this relation. Similarly, if the abstract *must* epistemic possibility relation were defined analogously to abstract *must* transition relations, it would not be necessarily symmetric. Therefore, we remove the abstract *must* epistemic possibility relations that are not symmetric. The labelling of an abstract state is defined so that it is consistent with the labelling of all the concrete states it represents. The bi-implication ensures that the abstract labelling function is *exact*.

We use an extension of the epistemic μ-calculus [20] as our specification language. We use the observational semantics for the epistemic component K_i in addition to the standard μ-calculus [21] and define the language $\mathcal{L}$ in BNF notation as follows. Let AP be a finite set of atomic propositions and $\mathcal{V}$ a set of propositional variables, then:

$$\varphi ::= \top \mid p \mid Z \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box\varphi \mid K_i\varphi \mid \mu Z.\varphi \mid \nu Z.\varphi$$

where $p \in AP$ and $Z \in \mathcal{V}$. Here $K_i\varphi$ means *agent* i *knows* φ [18].

The syntactic combinations μZ and νZ are the *least* and *greatest fix-point operators* respectively. An *interpretation* $\rho : \mathcal{V} \rightarrow 2^S$ assigns the free propositional variable Z as a set of states. Any occurrence of Z in φ falls within an even number of negations. Furthermore, we assume that formulas are *closed* and *well-named*, i.e., all propositional variables are bound exactly once in any formula.

To evaluate a formula φ, we compute sets of states such that a state s satisfies φ if $s \in [\![\varphi]\!]^{\mathcal{P},\rho}_{tt}$; a state s refutes φ if $s \in [\![\varphi]\!]^{\mathcal{P},\rho}_{ff}$. In addition to satisfaction (tt) and refutation (ff), we write $\perp$ to express that the truth value is unknown. We define the

three-valued semantics for $\mathcal{L}$ in line with [16] and extend it by the epistemic operator K_i as follows:

Definition 2 (Three-Valued Semantics). *Let $\mathcal{P}$ be AC-MAS. The three-valued semantics of $\varphi \in \mathcal{L}$ in $\mathcal{P}$ for an environment ρ, denoted $[\![\varphi]\!]_3^{M,\rho}$, is defined by a mapping $S \to \{\text{tt,ff}, \bot\}$ such that:*

$$[\![\varphi]\!]_3^{\mathcal{P},\rho}(s) = \begin{cases} \text{tt}, \textit{if } s \in [\![\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho} \\ \text{ff}, \textit{if } s \in [\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho} \\ \bot, \textit{otherwise} \end{cases}$$

The sets $[\![\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho} \subseteq S$ and $[\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho} \subseteq S$ for $\varphi \in \mathcal{L}$ over $\mathcal{P}$ are defined as:

$$\begin{aligned}
[\![\top]\!]_{\text{tt}}^{\mathcal{P},\rho} &= S & [\![\top]\!]_{\text{ff}}^{\mathcal{P},\rho} &= \emptyset \\
[\![p]\!]_{\text{tt}}^{\mathcal{P},\rho} &= \{s \in S : p \in \Lambda(s)\} & [\![p]\!]_{\text{ff}}^{\mathcal{P},\rho} &= \{s \in S : p \notin \Lambda(s)\} \\
[\![Z]\!]_{\text{tt}}^{\mathcal{P},\rho} &= \rho(Z) & [\![Z]\!]_{\text{ff}}^{\mathcal{P},\rho} &= \rho(Z) \\
[\![\neg\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho} &= [\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho} & [\![\neg\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho} &= [\![\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho} \\
[\![\varphi_1 \wedge \varphi_2]\!]_{\text{tt}}^{\mathcal{P},\rho} &= [\![\varphi_1]\!]_{\text{tt}}^{\mathcal{P},\rho} \cap [\![\varphi_2]\!]_{\text{tt}}^{\mathcal{P},\rho} & [\![\varphi_1 \wedge \varphi_2]\!]_{\text{ff}}^{\mathcal{P},\rho} &= [\![\varphi_1]\!]_{\text{ff}}^{\mathcal{P},\rho} \cup [\![\varphi_2]\!]_{\text{ff}}^{\mathcal{P},\rho} \\
[\![\Box\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho} &= ax([\![\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho}) & [\![\Box\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho} &= ex([\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho}) \\
[\![\mu Z.\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho} &= \text{lfp}(\lambda g.[\![\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho[Z \mapsto g]}) & [\![\mu Z.\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho} &= \text{gfp}(\lambda g.[\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho[Z \mapsto g]}) \\
[\![\nu Z.\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho} &= \text{gfp}(\lambda g.[\![\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho[Z \mapsto g]}) & [\![\nu Z.\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho} &= \text{lfp}(\lambda g.[\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho[Z \mapsto g]}) \\
[\![K_i\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho} &= ax_i([\![\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho}) & [\![K_i\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho} &= ex_i([\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho}) \cup [\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho}
\end{aligned}$$

where for $X \subseteq S$: $ax(X) = \{s \mid \forall s' : s \xrightarrow{\text{may}} s' \Rightarrow X\}$, $ex(X) = \{s \mid \exists s' : s \xrightarrow{\text{must}} s' \wedge X\}$, $ax_i(X) = \{s \mid \forall s' : s \overset{\text{may}}{\sim_i} s' \Rightarrow X\}$, and $ex_i(X) = \{s \mid \exists s' : s \overset{\text{must}}{\sim_i} s' \wedge X\}$. Intuitively, ax returns states whose may successors are all in X. In contrast, ex computes all states for which at least one must transition exists. Similarly, ax_i and ex_i are the corresponding operators for the epistemic relations for a given agent *i* and give the set of the respective indistinguishable states. The definition for $[\![K_i\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho}$ allows for a tighter under-approximation since agents do not know φ in states where φ is false.

An AC-MAS $\mathcal{P}$ satisfies a formula φ, or $[\mathcal{P} \overset{3}{\models} \varphi] = \text{tt}$, if all its initial states are in $[\![\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho}$. An AC-MAS $\mathcal{P}$ refutes φ, or $[\mathcal{P} \overset{3}{\models} \varphi] = \text{ff}$, if at least one initial state is in $[\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho}$. Otherwise we say $[\mathcal{P} \overset{3}{\models} \varphi] = \bot$. Note that the abstraction for AC-MAS models $\mathcal{P}$ as defined above is *consistent*, i.e., $[\![\varphi]\!]_{\text{tt}} \cap [\![\varphi]\!]_{\text{ff}} = \emptyset$ for any $\varphi \in \mathcal{L}$. Therefore the set $[\![\varphi]\!]_{\bot}^{\mathcal{P},\rho}$ can be computed as $S \backslash ([\![\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho} \cup [\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho})$.

Abstracting GSM. To instantiate the theory above, we now outline a methodology for constructing *abstract* AC-MAS models from *concrete* GSM programs. This process includes abstracting the data to build a finite model using *predicates*, as well as the computation of the temporal and epistemic *may* and *must* relations. Observe that GSM programs only regulate the evolution of the artifact-centric system in the presence of external events and do not include a description of the agents' behaviour with the system. To account for the evolution of both we combine GSM programs with procedural agent

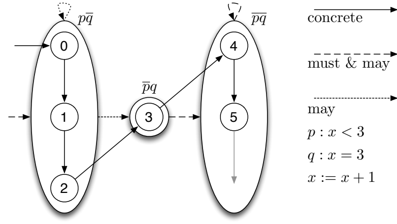

Fig. 2. Concrete and abstract transitions of a non-negative integer counter.

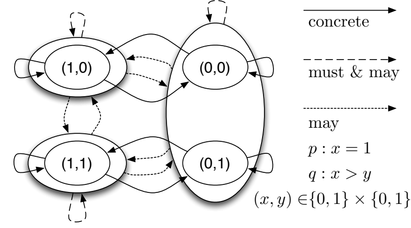

Fig. 3. Indistinguishable states of an agent given $y \in \nu$.

descriptions, thereby obtaining a GSM-MAS program. We do not present the agents descriptions here; we simply assume that they define the local states for the agents and define their evolution, both in terms of the actions performed on the artifact-centric system (or events) and the changes to their local state in the presence of actions. By GSM-MAS we refer to the combined programs consisting of the GSM code and the agents descriptions. It can be checked that AC-MAS provide a semantics for GSM-MAS programs.

Given a GSM-MAS program $\mathcal{P}$ and a specification φ as input, we generate an abstract $\mathcal{P}'$ such that if checking $\mathcal{P}' \models \varphi$ returns either *true* or *false*, then the same result also applies to $\mathcal{P}$; if $\mathcal{P}' \models \varphi$ returns undefined, then no conclusion can be drawn on $\mathcal{P}$ and the abstraction needs to be refined.

States in the abstract system are represented by *predicates*, which are Boolean variables that represent the validity of expressions in the concrete system. Predicates are selected by analysing the GSM-MAS program and the specification to be verified. In doing so we retain the status attributes of the lifecycles, as these are already Boolean, but replace the potentially unbound data attributes. To capture key conditions in the system, binary relations ($=, \neq, <, \leq, >, \geq$) or quantifications over sets of data ($\exists, \forall$) are selected by syntactically analysing the GSM-MAS program to get an initial set of predicates p_i.

In contrast to classical approaches, which build abstractions locally to single execution blocks, the declarative nature of GSM-MAS programs and the quantification over artifact instances results in predicates that are shared between instances or agents. While predicates that are *local* to an artifact instance or agent can be treated as instance variables, *shared* predicates need to be treated carefully to avoid incorrect abstractions for the local states of the agents. Building the abstract state using data predicates along with the original status attributes guarantees that the abstract system retains the same structure, while maintaining an over-approximation of the data space of the concrete system.

Since several concrete states correspond to an abstract state, temporal changes in the abstract system can only approximate the corresponding changes in the concrete data. Rather than giving the full procedure, instead we here compute the may and must transition relations on a simple example. Consider the abstraction of a non-negative integer counter with a single integer variable x that is initialised to 0 and gets incremented by 1 at each step using the assignment $x := x + 1$. If we base our abstract states on

the predicates $p : x < 3$ and $q : x = 3$, we have three possible abstract states, which are shown in Figure 2. Between the abstract states $p\overline{q}$ and $\overline{p}q$ we have a *may* transition because the concrete system *can* transition to a state that is in $\overline{p}q$. There is no *must* transition, however, because from a state in $p\overline{q}$ the concrete system can also transition to a state that is still in $p\overline{q}$. In contrast, all concrete states in $\overline{p}q$ transition to $\overline{pq}$, which means that we have both may and must transitions.

In line with existing literature in epistemic logic [18], the agents' knowledge is computed on the basis of the equality of their local components. In our case, however, the agents' local states are given by private variables, but also their *view* ν and the *window* ω. In the labelling algorithm for computing the sets in which an epistemic formula holds, the existential pre-image $\sim_i (X)$ of the set of global states X with respect to the appropriate epistemic relation ($\overset{\text{may}}{\sim}_i$ or $\overset{\text{must}}{\sim}_i$ is computed by existential quantification of variables outside of the view, and restriction to the window. The pre-image can be directly used to compute $[\![K_i\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho}$, since $\overset{\text{must}}{\sim}_i ([\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho}) = ex_i([\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho}) = \{s \mid \exists s' : s \overset{\text{must}}{\sim}_i s' \wedge [\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho}\}$. This is not the case for $[\![K_i\varphi]\!]_{\text{tt}}^{\mathcal{P},\rho}$, where $\overset{\text{may}}{\sim}_i (X) = \{s \mid \exists s' : s \overset{\text{may}}{\sim}_i s' \wedge X\}$; in this case we first compute the pre-image of $[\![\varphi]\!]_{\text{ff}}^{\mathcal{P},\rho}$ and then take its complement.

To build the abstract epistemic relations, views and windows have to be defined in terms of the predicates for the abstract states. The window ω can be expressed as a formula using relations between variables. Since we build our set of predicates using exactly those relations, we can build a direct mapping to an abstract function ω'. In other words, the abstract and concrete window functions represent the exact same states and $\omega(\gamma(x)) = \omega'(x)$ for any abstract state x.

The abstraction of the view ν is less straightforward, however, as predicates may use sets of variables that do not coincide with ν, and in the case of shared predicates may even relate to different instances and agents. This implies that an agent may be able to determine the value of a predicate only for some states. To avoid computing ν' depending on the state, we compute two sets ν_{may} and ν_{must} that give correct over- and under-approximations of the epistemic relation.

For the over-approximation $\overset{\text{may}}{\sim}_i$, we select only the *local* predicates for ν_{may} that exclusively refer to visible variables in ν. This ensures that an agent can distinguish two states in the abstract system only if it has enough visibility in the concrete system to determine the value of the predicates. We exclude *shared* predicates since one or more of the referenced instances might be outside the window ω and thus the predicate may be unknown. Note that fewer predicates in ν result in a larger set $\sim_i (X)$, thereby ensuring that an over-approximation is generated. This set is then restricted to the set R_{may} of reachable states computed with $\xrightarrow{\text{may}}$, which represent the states possibly reachable in the abstract model.

For the must transitions $\overset{\text{must}}{\sim}_i$, we need to ensure under-approximation; we stipulate that $s \overset{\text{must}}{\sim}_i t$ if for each of the concrete states in s there is a concrete state in t such that there is an epistemic relation for agent i between them. Intuitively, this means that we need to consider every predicate for ν_{must} that encodes at least one variable visible in the concrete system. Note, however, that this may not be sufficient as, if the predicates are not independent of each other, they may allow to infer information about a value even if it is not visible to the agent. Consider the example in Figure 3 with $p : x = 1$ and $q : x > y$ with the visible variable y. In the concrete system, $(x, y) = (1, 1)$ is

distinguishable from $(0,0)$, but not from $(0,1)$. To compute $\overset{\text{must}}{\sim_i}$ with visible predicate q and only quantify p would result in a transition between $p\overline{q}$ and $\overline{pq}$, which is not a proper under-approximation because of the missing epistemic relation between $(0,0)$ and $(1,1)$ in the concrete system. To ensure a correct under-approximation is generated, we transitively select all predicates that share the variables with predicates already in ν_{must} and also include *shared* predicates. Finally, we restrict $\overset{\text{must}}{\sim_i}$ by R_{must}, computed by $\xrightarrow{\text{must}}$, which corresponds to the set of states that are known to be reachable in the concrete system.

4 Implementation and Experimental Results

GSMC is an open source model checker that implements the technique described above [22]. It is operated via a command line application written in C++ that uses the `CUDD` library [23] for BDD operations and the SMT solver `CVC4` [24] to help compute the abstractions. `GSMC` uses binary decision diagrams (BDDs) to represent the sets of states and the transition relations of the abstract model.

`GSMC` operates directly on GSM programs developed in the `Acsi Hub` [4], a web-based application that supports the design and implementation of artifact systems. By using the `Acsi Hub`, users can design business artifacts with GSM lifecycles through a design editor and then immediately deploy these programs on an execution engine. The description of the agents and specification properties are supplied in plain text files.

`GSMC` supports specifications written in a temporal-epistemic logic with quantification over artifact instances. The language, called *Instance Quantified CTLK* [15], or IQ-CTLK, extends the usual epistemic branching time logic CTLK and has the following syntax:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid EG\varphi \mid E(\varphi U \varphi) \mid K_i\varphi \mid \forall x : R\, \varphi \mid \exists x : R\, \varphi$$

where R is the name of an artifact type and p is an atomic proposition over the agents' private data and the attributes of active instances that are specified in terms of *instance variables* bound by the quantification operators. The quantified instance variables range over the active instances of a given artifact type R in the state where the quantification is evaluated and must be bound.

We introduce a *bound* on the number of instances that can be generated and use an *overflow* flag that indicates if the bound was reached during a run. The bound in the number of instances restricts the possible behaviour of the system and may lead to loss of soundness or completeness when the limit is reached. The bound can be revised before any execution. Any IQ-CTLK formula to be verified is first rewritten into a CTLK formula by replacing the quantification operators as follows:

$$\forall x : \varphi \Rightarrow \bigwedge_{\iota \in \Gamma} created(\iota) \rightarrow \varphi \qquad\qquad \exists x : \varphi \Rightarrow \bigvee_{\iota \in \Gamma} created(\iota) \wedge \varphi$$

where the expression $created(\iota)$ checks if instance ι was created. This is required since the new formula ranges over the actual instances, which are created dynamically at run-time, and the number of *active* instances is not a priori known. The CTLK formula is

then translated to an epistemic μ-calculus formula using the fixed point characterisation of CTL [25]; the resulting specification is checked on the abstract model.

In the rest of the section we evaluate the tool. Both use cases are complete `Acsi Hub` applications. We verify the temporal-epistemic properties of the systems and discuss performance of the implemented techniques. All tests were conducted on a 64-bit Fedora 17 Linux machine with a 2.10GHz Intel Core i7 processor and 4GB RAM.

Evaluation: The Order-to-Cash scenario. This is an application in which a seller schedules the assembly of a product based on a confirmed purchase order from a buyer that requires several components, that are sourced from different suppliers. When the product is assembled, a carrier ships the order to the buyer. The buyer can cancel a purchase order at any time before the delivery. We refer to [17] for more details. The GSM program consists of a single-artifact `Acsi Hub` application with 10 data attributes, 9 stages, 11 milestones, and 12 events. We model a collection of components by introducing an integer counter. The process is considered complete when 3 components have arrived. The following three agent roles interact with the artifact system: 1) a *Buyer* who creates an artifact instance that represents the order; 2) a *Seller* who fulfils the order; and 3) a *Carrier* who ships the finished product to the Buyer.

We constructed several GSM-MAS with different numbers of agents and bounds on artifact instances. We report on the verification of these systems against four temporal-epistemic specifications. In the following *Diogenes* is an agent of role *Buyer*. The first specification, Property 1, states that *Diogenes* knows that the product might be be received via any of his orders as long as these are not cancelled, i.e., that there is no deadlock in processing the order:

$$AG\ \forall x : \mathit{CustomerOrder}((\mathit{x.BuyerId} \neq \mathit{Diogenes} \wedge \neg \mathit{Diogenes.Cancelled}) \rightarrow K_{\mathit{Diogenes}}\ EF\ \mathit{x.Received}) \quad (1)$$

Property 2 states that *Diogenes* may come to know that a product is received for an order with a different owner. This can be used to ascertain whether the orders are private to the buyers:

$$EF\ \exists x : \mathit{CustomerOrder}(\mathit{x.BuyerId} \neq \mathit{Diogenes} \wedge K_{\mathit{Diogenes}}\ \mathit{x.Received}) \quad (2)$$

Property 3 encodes the ability of an agent to deduce information it can not directly observe by checking whether *Diogenes* always knows there are 3 *PurchaseOrders* collected in all of his orders when the milestone *Ready* is achieved:

$$AG\ \forall x : \mathit{CustomerOrder}((\mathit{x.Ready} \wedge \mathit{x.BuyerId} = \mathit{Diogenes}) \rightarrow K_{\mathit{Diogenes}}\ (\mathit{x.PurchaseOrders} = 3)) \quad (3)$$

The last specification, Property 4, encodes the ownership of the order. It implies that an agent other than *Diogenes* can cancel an order that belongs to *Diogenes*. This is done by using a private variable, which is true only if *Diogenes* executed the *Cancelled* event. We thus require that an order that belongs to *Diogenes* cannot be cancelled if this variable is false:

$$EF\ \exists x : \mathit{CustomerOrder}(\mathit{x.BuyerId} = \mathit{Diogenes} \wedge \mathit{x.Cancelled} \wedge \mathit{Diogenes.cancelled} \neq 1) \quad (4)$$

Table 1. Performance for different numbers of artifact instances ι and agents.

	3 agents				15 agents			
$\#\iota$	#may	#must	MB	s	#may	#must	MB	s
1	0.91 e2	0.45 e2	55	0.2	1.65 e3	5.89 e2	69	2.1
2	2.23 e3	5.27 e2	78	0.9	1.32 e6	1.55 e5	106	4.6
3	5.34 e4	5.45 e3	93	4.8	1.03 e9	3.83 e7	124	31.9
4	1.28 e6	5.46 e4	112	25.5	7.99 e11	9.02 e9	233	168.8
5	3.10 e7	5.42 e5	172	90.4	6.05 e14	2.05 e12	463	596.2
6	7.57 e8	5.36 e6	273	257.2	4.53 e17	4.57 e14	898	2014.2

We first verified the properties in the abstract system and measured the number of may and must reachable states, memory used, and CPU time required. GSMC evaluated Property 1 to be *unknown*, Properties 2 and 4 to be *false*, and Property 3 to be *true* in the abstract model. Table 1 reports the performance for a system with 1 agent per role and a system of 15 agents (6 Buyers, 5 Sellers, and 4 Carriers). We observe that there is an order of magnitude of difference in the number of may and must reachable states; this implies that there are specifications, such as Property 1, that cannot be determined. However, the tool is still able to find answers to the other three properties. The results are in line with our expectations, confirming the correctness of the GSM program against said specifications.

For a comparison we disabled the predicate abstraction feature and verified the same Order-to-Cash system under the same conditions. In this case GSMC evaluated Properties 1 and 3 to be *true* and Properties 2 and 4 to be *false* in the model, which is consistent with the abstraction results. Note that the previously unknown Property 1 is returned as *true* when predicate abstraction is disabled.

Table 2 presents the performance of the tool executed on the same machine, under the same conditions. By comparing this table to Table 1, we see that verification of the concrete model initially outperforms abstraction. This is because there is a constant overhead from building the may and must temporal transitions by calls to the SMT solver. However, as the model grows we clearly see the benefits of the abstraction methodology as it reduces the number of states to be considered. For example, for 15 agents and 5 instances we have over two orders of magnitude reduction in the number of states to be considered and an order of magnitude reduction in the verification time.

Although the tool does not support automatic refinement for the abstraction methodology, by manually adding the predicates $x.PurchaseOrders = 0$, $x.PurchaseOrders = 1$, and $x.PurchaseOrders = 2$ we could refine the abstract model in such a way that may and must reachable state spaces become equal to those of the concrete model. In doing so Property 1 is no longer returned as unknown but true; this is in line with the results obtained by verifying the concrete system.

The second evaluation scenario focuses on the management of research programs.

The scenario consists of three conceptual entities modelled as business artifacts: *CallForProposals* represents the annual call of a funding program; *Project* encodes one

Table 2. Performance for different settings of the concrete system.

	3 agents			15 agents		
$\#\iota$	#states	MB	s	#states	MB	s
1	1.17 e2	27	0.1	2.92 e3	31	0.2
2	3.71 e3	52	0.7	4.16 e6	70	4.9
3	1.16 e5	64	5.9	5.82 e9	84	65.5
4	3.67 e6	96	42.1	8.01 e12	222	360.2
5	1.18 e8	195	176.7	1.09 e16	539	1419.6
6	3.83 e9	375	500.5	N/A	N/A	N/A

project which starts as a proposal and, if successful, becomes a funded research project; *ReviewBoard* governs the assembling of a review board for a specified research topic and the reviews of all competing proposals. We focus on three roles: the *Program Manager* initiates the process and confirms the board; the *Program Staff Member* supervises projects on behalf of the funding agency, the *Project Leader* is responsible for a particular proposal. The scenario was implemented in the `Acsi Hub`. We refer to [26] for detail.

The GSM program for this scenario is a significantly larger application than the Order-to-Cash, as it consists of 45 stages, 56 milestones, and 19 events. For this reason we here report only the interactions between the agents and the *ReviewBoard* artifact type only, i.e., the types *CallForProposals* and *Project* are not analysed here. We also restrict the number of agents to one per role. Nevertheless, GSMC builds the transition relations for the *whole* GSM program.

An artifact instance is created when the agent *Manager* decides to set up a review board. When the *Manager* confirms the assembled board, the lifecycle of the *ReviewBoard* instance terminates. The agent *Staff* carries out several administration task, including assembling and updating the review board. Both *Manager* and *Staff* can access all artifact instances. In contrast, the agent *Leader* cannot observe any of them. Agents do not set specific payloads; this implies we can examine all the possible non-deterministic behaviours.

The first two specifications we analyse concern the simple reachability of stages and milestones. Property 5 states that there is an instance of the *ReviewBoard* artifact type in which eventually the stage *SendProposalsToReviewers* is open:

$$EF\ \exists x : \mathit{ReviewBoard}(x.\mathit{SendProposalsToReviewers}) \tag{5}$$

Property 6 encodes that there is an instance of *ReviewBoard* in which eventually the milestone *ReviewsTerminated* is achieved. This means that an instance will terminate:

$$EF\ \exists x : \mathit{ReviewBoard}(x.\mathit{ReviewsTerminated}) \tag{6}$$

The next two specifications demonstrate the use of 3-valued abstraction on sets of data. These formulas cannot be verified on concrete systems as sets of data cannot be represented on concrete models. Property 7 states that there is an instance of *ReviewBoard*

Table 3. Performance results for 1 instance of the *ReviewBoard* artifact type.

Operation	Result	Memory	Time
Computation of τ_m and τ_M	✓	395MB	33.16s
Computation of R_{may} and R_{must}	✓	364MB	3.06s
Property 5	✓	280MB	1.21s
Property 6	✓	284MB	1.02s
Property 7	✓	278MB	0.80s
Property 8	✓	272MB	0.92s
Property 9	✓	312MB	1.54s
Property 10	✗	320MB	2.22s

in which eventually the the active reviewers is equal to the specified number of reviewers required:

$$EF\ \exists x : \mathit{ReviewBoard}(x.\mathit{Reviewers.size}() = x.\mathit{ReviewBoardSize}) \tag{7}$$

Property 8 states that there is an instance of *ReviewBoard* in which eventually the set of active reviewers contains a reviewer called *Diogenes*:

$$EF\ \exists x : \mathit{ReviewBoard}(x.\mathit{Reviewers.exists}(\mathit{FirstName} = \mathit{Diogenes})) \tag{8}$$

The last two specifications concern reasoning about the knowledge of the agents. Property 9 says that agent *Manager* knows there is a path where eventually the milestone *ReviewsTerminated* is achieved:

$$K_{\mathit{Manager}}\ (EF\ \exists x : \mathit{ReviewBoard}(x.\mathit{ReviewsTerminated})) \tag{9}$$

Finally, Property 10 encodes that agent *Leader* knows there is a path where eventually the milestone *ReviewsTerminated* is achieved:

$$K_{\mathit{Leader}}\ (EF\ \exists x : \mathit{ReviewBoard}(x.\mathit{ReviewsTerminated})) \tag{10}$$

The data attributes of the concrete model are represented by 10 predicates in the abstract model. The abstract model is then encoded by GSMC into BDDs by using 142 Boolean variables. As the construction of the transition relations requires three distinct sets of Boolean variables, there are 426 Boolean variables in total. The *may* reachable state space of the model spans over approximately 7.1×10^9 states, and its construction requires 30 iterations. The *must* reachable state space has 8.4×10^7 states and it is built in 12 iterations. The total time for the verification was 43.88s and the memory usage peaked at 395MB.

Table 3 presents the performance of the individual operations undertaken by GSMC, as well as the verification results. The first row reports the construction of the transition relations, the second row shows the construction of may and must reachable state spaces, and the remaining rows give the performance for the properties verified in this section. Properties 5 to 9 are *true* in the model. Property 10 is *false* in the model since the agent *Leader* cannot observe the *ReviewBoard* lifecycle.

5 Conclusions

Artifact-centric systems have been put forward as an intuitive paradigm to model applications for businesses and services. Differently from process models, artifact-centric systems give equal prominence to both the process model (i.e., the lifecycles) and that information model (i.e., the data structures). GSM has been introduced as a programming framework for artifact-centric systems and recently adopted as part of the OMG Case Management Model and Notation standard [27]. This suggests its use may increase considerably in the future.

In this paper we introduced a methodology for the verification of GSM systems. The technique extends state-of-the-art methods in verification by providing a predicate abstraction methodology to GSM. In addition to catering for GSM programs directly, we support first-order quantification to refer to the data referenced by artifacts. Differently from any other mainstream predicate abstraction technique we also support operators expressing the knowledge of the agents in the system.

We implemented the technique in GSMC, the first model checker for GSM that supports GSM's information model. The checker supports GSM's infinite models and automatically generates, via SMT calls, finite abstract models that can be efficiently encoded as BDDs and then verified. To evaluate the efficiency of the approach we have discussed the experimental results obtained by using the checkers on sophisticated use-cases generated by third-parties in the EU project ACSI. The approach as currently implemented does not support recursion in the GSM programs. In the future we plan to add partial support for basic recursive data types and automatic refinement.

References

1. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering **32**(3) (2009) 3–9
2. Marin, M., Hull, R., Vaculín, R.: Data Centric BPM and the Emerging Case Management Standard: A Short Survey. In: Proceedings of Business Process Management Workshops - BPM 2012 International Workshops. Revised Papers. Volume 132 of Lecture Notes in Business Information Processing., Springer (2013) 24–30
3. Heath, F.T., Hull, R., Vaculin, R.: Barcelona: A design and runtime environment for modeling and execution of artifact-centric business processes. Demo track in International Conference on Business Process Management 2011 (2011)
4. Boaz, D., Heath, T., Gupta, M., Limonad, L., Sun, Y., Hull, R., Vaculín, R.: The ACSI hub: A data-centric environment for service interoperation. In: Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014). Volume 1295 of CEUR Workshop Proceedings., CEUR-WS.org (2014)
5. Baresi, L., Bianculli, D., Ghezzi, C., Guinea, S., Spoletini, P.: Validation of Web Service Compositions. IET Software **1**(6) (2007) 219–232
6. Alonso, G., Casati, F., Kuno, H.A., Machiraju, V.: Web Services - Concepts, Architectures and Applications. Data-Centric Systems and Applications. Springer (2004)
7. Graf, S., Saidi, H.: Construction of abstract state graphs with pvs. In: Computer Aided Verification. Volume 1254 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1997) 72–83

8. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of Deployed Artifact Systems via Data Abstraction. In: Proceedings of the 9th International Conference on Service-Oriented Computing (ICSOC'11). Volume 7084 of Lecture Notes in Computer Science., Springer (2011) 142–156
9. Belardinelli, F., Lomuscio, A., Patrizi, F.: An abstraction technique for the verification of artifact-centric systems. In: Proceedings of Principles of Knowledge Representation and Reasoning (KR'12). (2012) 319–328
10. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of GSM-based artifact-centric systems through finite abstraction. In: Proceedings of the 10th International Conference on Service-Oriented Computing (ICSOC'12). Volume 7636 of Lecture Notes in Computer Science., Springer (2012) 17–31
11. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: Proceedings of the 5th International Conference on Business Process Management (BPM'07). Volume 4714 of Lecture Notes in Computer Science., Springer (2007) 288–304
12. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic Verification of Data-centric Business Processes. In: Proceedings of the 12th International Conference on Database Theory (ICDT'09). Volume 361 of ACM International Conference Proceeding Series., ACM (2009) 252–267
13. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of agent-based artifact systems. Journal of Artificial Intelligence Research **51** (2014) 333–376
14. Gonzalez, P., Griesmayer, A., Lomuscio, A.: Verifying GSM-based business artifacts. In: Proceedings of the IEEE International Conference on Web Services (ICWS'12). (2012) 25–32
15. Gonzalez, P., Griesmayer, A., Lomuscio, A.: Model checking GSM-based multi-agent systems. In: Service-Oriented Computing–ICSOC 2013 Workshops. (2013)
16. Shoham, S., Grumberg, O.: 3-valued abstraction: More precision at less cost. Information and Computation **206**(11) (2008) 1313 – 1333
17. Hull, R., et al.: Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In: Proceedings of the 5th ACM International Conference on Distributed Event-Based Systems (DEBS'11). (2011)
18. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. The MIT Press (1995)
19. Chechik, M., Devereux, B., Easterbrook, S., Gurfinkel, A.: Multi-valued symbolic model-checking. ACM Transactions on Software Engineering and Methodology **12**(4) (2003) 371–408
20. Bozianu, R., Dima, C., Enea, C.: Model-checking an epistemic μ-calculus with synchronous and perfect recall semantics. CoRR **abs/1310.6434** (2013)
21. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science **27**(3) (1983) 333 – 354
22. Gonzalez, P., Griesmayer, A., Lomuscio, A.: GSMC: A model checker for GSM. http://http://vas.doc.ic.ac.uk/software/extensions/ (2014)
23. Somenzi, F.: CUDD: CU Decision Diagram Package Release 2.5.0. (2012) `http://vlsi.colorado.edu/~fabio/CUDD/`.
24. Barrett, C., Tinelli, C.: CVC4 Version 1.2. (2013) `http://cvc4.cs.nyu.edu/web/`.
25. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press (2004)
26. Toribio Gomez, D., Murphy-O'Connor, C., De Leenheer, P., Malarme, P.: D5.5 Deployment and evaluation of pilots using final ACSI hub system results and evaluation. Project deliverable, The ACSI Project (EU FP7-ICT-257593) (2013)
27. Group, O.M.: Case management model and notation, ver 1.0. Technical report (2014)