ACSI – Artifact-Centric Service Interoperation

D2.1.1
Artifact verification and synthesis techniques
Iteration 1

Project Acronym	ACSI	
Project Title	Artifact-Centric Service Interoperation	
Project Number	257593	
Workpackage	2 Formal-based techniques and tools	
Lead Beneficiary	Imperial College of Science, Technology and Medicine	
Editor(s)	Francesco Belardinelli	Imperial
	Alessio Lomuscio	Imperial
	Fabio Patrizi	Imperial
Contributors(s)	Ioana Boureanu	Imperial
	Giuseppe De Giacomo	Uniroma
	Riccardo De Masellis	Uniroma
	Paolo Felli	Uniroma
	Babak Bagheri Hariri	Bolzano
	Diego Calvanese	Bolzano
Reviewer	Dirk Fahland	TU/e
Dissemination Level		Public
Contractual Delivery Date	30/05/2011	
Actual Delivery Date	30/05/2011	
Version	0.4	

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 257593

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 257593

Abstract

This document presents the achievements of Task 2.1 "Artifact Verification and Synthesis Techniques – Iteration 1". We describe the requirements to be fulfilled in order to develop effective procedures for the verification and synthesis of artifact systems, then review related work, present two original contributions, and discuss our future plan.

Document History

Version	Date	Comments
0.1	30-03-2011	Document structure and ToC
0.2	02-05-2011	First draft
0.3	15-05-2011	Submission for internal review
0.4	30-05-2011	Final version

Table of Contents

List of Tables

List of Figures

Acronyms

Acronym	Explanation
FO	First Order
WP	Work Package
MC	Model Checking
MAS	Multi-Agent System
ACS	Artifact-Centric System
QIS	Quantified Interpreted System

1 Introduction

This document describes the achievements of Task 2.1 (T2.1) "Artifact Verification and Synthesis Techniques" of Work Package 2 (WP2), after one year from the beginning of the ACSI project. The ultimate goal of the task is the definition of techniques for the verification and synthesis of artifact systems.

By verification we mean that given a formal model describing (a suitable abstraction) of the evolution of an artifact system, one is able to check whether some properties of interest are satisfied by the model (and hence the system it represents). Properties of interest span from simple ones like reachability of a particular state, e.g., checking if the system can reach a state where no artifact is present, to more complex ones, like checking that each time an artifact is added, it will eventually reach a particular status, or that the data content of the system at each step always satisfies some (relational) constraints, or even epistemic properties requiring that a participant has no strategy to access some "sensitive" information.

By synthesis, generally speaking, we mean the process of (automatically) constructing coordinating mechanisms that make the artifacts of an artifact system to interoperate so as to guarantee satisfaction of properties similar to those mentioned above. This document discusses some work and issues related to the synthesis task, but no results are provided so far. This is due, in particular, to the fact that we expect, similarly to the simpler propositional case, verification and synthesis techniques to share the very same theoretical framework, as well as verification techniques to provide useful guidelines for the development of synthesis procedures. Therefore, a good understanding of verification in the artifact context can be seen as a *prerequisite* necessary to tackle the harder issue of synthesis.

Two fundamental tasks are required in order to accomplish T2.1. Namely, formal models for the artifact systems need to be developed, and corresponding formalisms for the specification of the properties of interest need to be adopted. Of course, we expect a tradeoff between richness of the model and expressivity of the specification formalism, and the complexity of the verification techniques. That is, the closer the model to the actual artifact system and the more expressive the specification formalism, the harder the algorithm to check the properties against the model.

Having in mind the objective of T2.1, our efforts have focused on identifying promising models and formalisms to capture interesting verification problems, as well as on investigating their computational properties, in terms of decidability and complexity. In doing so, we have taken into consideration the outcome of WP1, where, in particular, a general, abstract, model has been proposed for artifact systems, so as to come up with (possibly simplified) models that are compliant with the general structure proposed by WP1.

The distinguishing feature of the model developed in WP1, as largely anticipated in preliminary analyses, is the presence of data and its infiniteness, which makes the systems to deal with infinite-state. This clearly constitutes a major obstacle to both verification and synthesis, as non-trivial problems easily become undecidable. This feature and the fact that the data organization is an aspect of interest –e.g., one may be interested in whether each artifact instance, at some point, has a name and is associated to at least another artifact instance– are the main ingredients that make the verification and synthesis problems challenging, as current results do not provide actual techniques that can be successfully employed *off-the-shelf*, nor easily adapted to solve them.

We start by identifying a core of requirements to fulfill in order to develop a framework and corresponding techniques for formal verification and synthesis in this context (Section 2). This is done by extending the typical requirements of standard (propositional) verification of finite-state

systems to the case of infinite-state systems with data, essentially relying on previous work (cf., e.g., [CGP99, DSV07]) and past experience [LQR09].

Once the requirements have been identified, we review related work that can serve as a basis for their fulfillment (Section 3). In particular, we discuss works on verification and synthesis of finite- and infinite-state systems, and on abstraction techniques that can be useful in both narrowing the search space (thus providing a further solution to the *state-explosion* problem) and reducing a verification problem from infinite- to finite-state, thus guaranteeing decidability. Moreover, we briefly discuss the critical points in our project that cannot be directly tackled by the existing techniques.

Then, we describe our main contributions. Firstly (Section 4), we adapted a framework for Model Checking of Multi-Agent Systems, namely that of *Quantified Interpreted Systems* [BL09], to the context of artifact systems. Quantified interpreted systems constitute a natural framework for reasoning about the evolution of an artifact system, as well as about epistemic properties involving the participating participant. However, while certainly relevant from a theoretical point of view, their original formulation turns out too general, and possibly more powerful than needed for our purposes, to guarantee decidability of non-trivial verification tasks. In addition to detailing the framework, the work shows how to use it to capture verification problems on artifact (infinite-state) systems, and provides a first classification of abstraction approaches, isolating, in particular, a case of practical interest where the problem can be reduced to finite-state (hence decidable) Model Checking.

The second contribution (Section 5) focuses on verification of temporal properties over data, without epistemic modalities. It develops a technique to detect if an artifact system, specified by its data schema and exposed actions, is guaranteed to deal with a bounded number of new elements, when a given process is executed over it. When this is the case, the verification problem reduces, again, to the finite case and can hence be addressed by means of (possibly adapted) standard Model Checking techniques. Importantly, this work does not focus on a particular temporal fragment, but considers the general μ-calculus over *relational states*.

Finally, we draw conclusions about the work carried out so far, and analyze future developments.

2 Requirements

In this section, we discuss the requirements that our framework needs to fulfill in order to serve as a formal basis in the development of actual tools for verification and synthesis of artifact(-based) systems.

Verification of artifact systems refers to the process of checking whether a given system satisfies some design requirements. Such requirements can be broadly divided into two classes:

Intra-artifact requirements, capturing properties related to the internal structure of a particular artifact instance. They can refer to the current state of the artifact instance, e.g., requiring that every value of a particular attribute in an instance refers to another attribute in the same artifact, or can be more complex and take into account the whole instance lifecycle, requiring, e.g., that each time an artifact is created, it is eventually destroyed.

all along its evolution, the artifact instance goes through some particular state infinitely often.

Inter-artifact requirements. More general than those above, requirements from this class

capture properties involving the interaction among artifact instances. An example is checking that every sent message is eventually consumed, or that artifact instances of a particular type are indeed generated by some instance of another type.

Conformance checking is the task of verifying an artifact system against a set of intra- and inter-artifact requirements, meant to capture the intended lifecycle of artifact instances. This is of particular interest as, when successful, guarantees that a system actually behaves as intended. Indeed, one of the main goals of this WP is to *build a formal framework providing the theoretical foundations and the algorithmic support for conformance checking.* This suggests, as a first requirement, that our framework must provide:

R1 *appropriate formalisms and techniques able to capture and verify temporal properties.*

However, this is not enough. As it turns out from WP1, the states of an artifact system can be modeled as (sets of) relational database instances, whose evolution essentially depends on updates performed by external participants, often human operators. As a consequence, no assumption can be made, in general, on the domains of the new values introduced at each step, and, in particular, such domains can be infinite, thus giving raise to infinitely many (relational) states. Of course, such features need to be accounted for in our framework, which thus requires that:

R2 *the formalism for requirement specification be able to express relationships among data*;

R3 *the verification techniques have the ability to deal with some form of infinite-state systems, possibly through approximations to finite models.*

As to [R2], a natural choice is to consider specification languages that include appropriate fragments of first-order logic. Concerning [R3], that is clearly affected by the (expressive power of) the language above, it is clear that classical techniques developed for finite-state systems cannot be applied, in general. Moreover, it turns out that all non-trivial, sufficiently general problem formalizations are undecidable, so that no complete verification procedure can be developed for the general case. Approaches to address this issue include: restricting the class of requirements to verify, restricting the class of systems, or combinations of both. Under such restrictions, it is often possible carrying out an *abstraction* process to produce a finite representation, the *abstract system*, of the original, concrete, infinite system, that can then be processed using classical techniques for the finite case, so as to obtain relevant information on the infinite behavior of the concrete system (cf., e.g., [BCG$^+$05, DSV07, DHPV09, DDV11]). This is the approach currently explored in the ACSI project, although others, in particular incomplete techniques that do not necessarily resort to abstraction, are certainly worth investigating.

We stress that dealing with infinite states is the main challenge for the verification and synthesis tasks. Similarly to previous work (cf., e.g., Section 3), we expect to be able to develop effective techniques only under appropriate restrictions, or by relaxing some requirements (e.g., completeness) on the quality of the techniques. Nonetheless, we believe that even under such restrictions procedures able to solve interesting cases can be developed.

Observe that there is no reason, in general, for restricting verification to conformance checking only. Indeed, once a framework for verification of intra- and inter-artifact requirements is set, it can be used for the verification of any requirement, even if not (directly) related to artifact lifecycles. Therefore, we refer to *verification* as the generic task of checking whether an artifact-system fulfills a given set of (inter- and/or intra-artifact) requirements, which includes conformance checking as a special case.

In addition to verifying natural requirements as those described above, which essentially concern dynamic properties internal to the system, we also want to take a more external perspective, and verify requirements that involve the knowledge each participant has about the system, and in particular the data content. Such an interest is motivated by the observation that, in many cases, access to artifact data may be restricted, e.g., for security reasons, so a participant may be able to access only a partial fragment of the data content, and this might affect its ability to carry out (some of) the operations it was originally expected to offer. For instance, if a human participant, in order to serve a two-way call, needs to send an e-mail to someone whose address is recorded in an artifact inaccessible to the participant, then the required operation cannot be performed, and this may result in an undesired evolution of the artifact that initiated the two-way call (that, possibly, considers the call unsuccessful, after a timeout has occurred). But also more involved requirements are conceivable. Assume an artifact system is designed to prevent a certain class of participants from accessing some information. Verifying that the design actually enforces this property is certainly of interest, and this calls for checking the requirement that participants of the "bad" class cannot act jointly to retrieve (any fragment of) the reserved information.

In order to allow for verifying such *epistemic* requirements, our theoretical framework needs to be able to express knowledge-related properties. Therefore, it needs:

R4 *a formalism for requirement specification able to express epistemic properties involving data*;

R5 *verification technique(s) able to verify (an interesting class of) epistemic properties expressible in the formalism above.*

As it turns out, [R4] is an extension of [R2], as epistemic properties refer to (relationships among) data, and, similarly, also the corresponding verification techniques ([R5]) extend [R4]. Observe, however, that verification procedures tailored on special classes of requirements, often more effective than general ones, can be developed.

Importantly, as the objective of this WP is the development of actual, effective techniques for the verification of artifact systems, it seems natural to consider formalisms for requirement specification that are *computationally grounded* [Woo00] on existing models –such as the abstract model developed in WP1– that are well-suited to describe the (possibly simplified) evolution of artifact systems. This essentially means that the semantics of formulas in the adopted language must be provided in terms of such concrete (computational) models.

R6 *The language for requirement specification must be computationally grounded on existing computational models of artifact systems.*

In particular, observe that when considering epistemic properties, the notion of *agent* (capturing participants that interact with the artifact system) is an indispensable part of the model, as it represents the subject of knowledge, while it can certainly be removed if knowledge-related properties are not of interest.

Finally, we aim at actually implementing the techniques developed in this WP. In order for instances of the verification problem to be encoded and provided in input to the verification tool,

R7 *both computational models and requirement formalisms must be amenable to actual modelling languages of future design.*

In this respect, a suitable candidate language for models seems to be ISPL (Interpreted Systems Programming Language, see [LQR09]), possibly enriched with first-order features.

As to synthesis in the context of artifact systems, broadly speaking, it refers to the definition of control procedures able to interact with the clients of the artifact system, and to control the system so as to guarantee desired requirements. Such requirements typically include properties about the data content of the system, its evolution, and even the epistemic state of participants.

For instance, one may need to build a *mediator* process that, by interacting, on the one side, with the artifact system, provides, on the other side, participant with a customized access to the system, e.g., by enabling only a limited set of actions on artifacts, or restricting the access to some artifact's attribute. In other words, such a process would act as a *filter*.

Typically, synthesis is harder than verification. In the literature a notable example of such a gap is the case of linear-time temporal logic, where the verification problem is PSPACE-complete [CGP99], while the synthesis problem for the same logic has a doubly exponential bound [PR89] (both in the size of the requirement specification). These problems have a tight connection in that synthesizing a system for a given requirement is successful if and only if the obtained system can be successfully verified against the requirement. Intuitively, synthesis can be seen as preparing a system for a successful verification test, thus suggesting that, similarly to the case of synthesis for temporal specifications, in order to actually devise effective synthesis techniques, we need that

R8 *the framework and techniques developed for formal verification should be amenable for use/extension to synthesis.*

Observe that this includes the formalism for requirement specification and its respective semantics, defined according to the requirements identified above. As a desirable side-effect, we expect verification techniques to provide useful guidelines on the development of synthesis procedures.

3 State of the art

In this section, we review the existing contributions in formal verification, synthesis, and abstraction literature, that are relevant to ACSI. Moreover, we discuss the main ACSI requirements that make such achievements not fully applicable in our context. As it often happens, even if the existing results do not perfectly match all project requirements, thus are not straightforwardly applicable, they certainly represent a solid background for solving problems that arise from specific project needs.

3.1 Verification

Approaches to formal verification can be broadly divided into two categories, depending on whether the class of the systems verified is finite- or infinite-state. In the following we review the previous contributions that are most relevant to ACSI.

3.1.1 Finite-state systems

By *finite-state* systems we mean (real) systems with a finite, (possibly huge) space of possible configurations. Typical examples of such systems include traffic lights, communication protocols, and hardware systems. One of the most popular techniques for verification of finite-state systems is *model checking* [CGP99], a technique based on building a suitable mathematical description

of the system, referred to as *model*, of the requirement to verify, i.e., the *specification*, and then executing a verification algorithm that checks whether the model satisfies the specification.

Formally, the model is described as a *Kripke structure*, or *transition system*, i.e., essentially, a graph whose nodes represent different system states and whose edges capture possible state transitions. Each state is associated with a finite set of *propositional labels*, each representing an atomic property satisfied by the current state. For instance, if in the traffic-light example one represents the colors of the lights currently displayed by means of propositions r(ed), y(ellow) and g(reen), then the state where the red light is displayed would be labelled by r, while the state where yellow and green are displayed at once would be labelled by y and r.

As for specifications, they are typically formulas expressed in (fragments of) the *computation tree logic* CTL* [CES86, EH86], a language that, interpreted on Kripke structures, allows for capturing temporal possibility and necessity of properties to happen over time, as the system evolves. For instance, in the traffic-light example, one may express properties such as "it is always the case that the red light is eventually displayed", or "when the green light is displayed, the yellow light is displayed next".

Despite their relative simplicity, CTL* and its most widely used fragments, the *branching-time* temporal logic CTL [BAPM83, EC80] and the *linear-time* temporal logic LTL [Pnu81], proved successful in capturing many specifications of interest in the practice, as witnessed by their affirmation in industrial contexts for verification of both hardware and software (real) system components (see, e.g., [CW96] for a list of notable examples). More general specifications of Kripke structures can be expressed in propositional μ-calculus [Koz83], a modal logic with least and greatest fixpoint operators, powerful enough to capture the whole CTL* [Dam94].

As one may expect, depending on the expressive power of the specification language, one gets different complexity bounds of the verification task. In particular, verification of CTL formulas is polynomial in the size of the Kripke structure and the formula [CES86], LTL verification is PSPACE-complete [SC85], and so is CTL* [SC85]. For both LTL and CTL*, the best known algorithms run in exponential time wrt the size of the requirement specification, while for propositional μ-calculus, the best known algorithm runs in exponential time in the *alternation depth* [1] of the specification [BCJ$^+$97], and polynomial time in the size of the Kripke structure.

Typically, the model of a system (i.e., the Kripke structure) is not provided explicitly, i.e., listing all of its nodes, transitions and labels, but in a compact way, using a suitable language that, when interpreted, *generates* the Kripke structure. By *compact*, we mean that the size of the representation, also referred to as the *program*, is polynomial in the number of propositions labeling the states.

However, the underlying Kripke structure may contain a number of states that is exponential in that of propositions. The main problem associated with this phenomenon is known as the *state explosion* problem. Due to this, the explicit search over the whole set of states represented explicitly makes (explicit) model checking unsuitable for many real systems that, although finite, are characterized by a huge number of states.

The great success of model checking is due to the existence of optimization techniques that make the verification algorithms practically effective. Some of these are based on representing the underlying Kripke structure in a *symbolic* way, by using *Ordered Binary Decision Diagrams* (OBDD) [Bry86], i.e., an efficient representation of Boolean formulas used to characterize states and transitions of the Kripke structure [BCM$^+$92]. Such optimizations made possible the implementation of verification tools, known as *symbolic* model checkers, able to manipulate

[1] A measure depending on the syntactic structure of the formula, which reflects the number of nested fixpoints that need to be computed in order to verify the formula.

models large enough to capture real systems, and consequently allowing for formal verification of such systems. Examples of symbolic model checkers include SMV [2], NuSMV [3], CadenceSMV [4].

Other optimization strategies are based on *partial order reduction*, a technique that allows for performing the search only on a *representative* subset of the state space, by exploiting the fact that the execution order of some transitions leads to states that are equivalent with respect to the requirement under verification. Typically, such techniques are best suited for *concurrent* systems verification. *Explicit* model checkers include SPIN [5], UPPAAL [6], and LoLA [7].

An experimental comparison of the performance of various tools in the verification of LTL specifications was carried out in [RV10], showing that, on such class of specifications, symbolic tools significantly outperform explicit ones.

So far, we considered verification and model checkers for temporal specifications, only. However, the verification approach based on checking a model against a specification applies in general to other contexts, as well. Of particular interest to ACSI is *verification of multi-agent systems* (MAS) [Woo09] or, more specifically, model checking for *temporal-epistemic* logics [vdMS99, KLP04]. This essentially refers to verifying whether some agents, acting and interacting according to their *protocols*, satisfy a desired specification. Such specifications, differently from the ones above, not only express properties concerning the system's evolution, but also allow for expressing the knowledge state, whereby *epistemic*, of agents with respect to both other agents and the environment they act in, as well as the ability of agent *coalitions* to achieve temporal-epistemic goals. Examples include: "an agent knows that another agent knows something", "all agents know that a particular property holds at a given state", "agents 1, 2, and 3, can cooperate so as to guarantee a particular property to be always satisfied", "agents 1 and 2 can prevent agent 3 from knowing what they know".

As it turns out, the language for such specifications needs, of course, appropriate constructs, including temporal operators such as those found in CTL*, *cooperation modalities* [AHK02] that allow for considering agent coalitions, and epistemic operators [FHMV95] to express properties about agents' and coalitions' state of knowledge. Many languages can be used, depending on the class of specifications one needs to capture. Of course, generality has a cost. For instance, the well-known *Alternating-time temporal logic* (ATL*) [AHK02], which provides temporal operators and cooperation modalities, but no epistemic operators, is a very rich language, strictly more expressive than CTL*, and, as expected, computationally more demanding. Precisely, ATL* model checking is a 2EXPTIME-complete problem [AHK02]. For this reason, the ATL* fragment ATL, where temporal operators are required to occur paired with a cooperation modality, is often preferred, as it provides a good compromise between expressivity and complexity. Indeed, ATL model checking is PTIME-complete [AHK02] (specifically, polynomial in the size of the structure to verify and the size of the specification) thus "only" exponential in the compact description of the problem instance. This suggests that the specification language needs to be carefully chosen, so as to obtain enough expressive power to capture (most of) the specifications of interest, while guaranteeing feasibility of the verification task.

The relevance of MAS verification, and in particular model checking, comes essentially from two observations. Firstly, tools and techniques for MAS model checking have been successfully

[2] `www.cs.cmu.edu/~modelcheck/smv.html`
[3] `nusmv.fbk.eu/`
[4] `www.kenmcmil.com/smv.html`
[5] `spinroot.com`
[6] `www.uppaal.com`
[7] http://www.informatik.uni-rostock.de/tpp/lola/

employed in the verification of several scenarios involving services as basic components [LQS08a, LQSS07, LQS08b], thus showing that multi-agent frameworks can be successfully adopted to capture service-oriented architectures. Secondly, agents can be used to model the interaction among the stakeholders and the artifact system, thus allowing for the verification of properties concerning not only the state of the system, but also the behavior of each stakeholder, along with its epistemic state. For instance, one may check whether a stakeholder can access some information provided by the system, or if it can cooperate with other stakeholders so as to force a particular evolution of the system.

In this project, we focus our attention on the MAS verification tool MCMAS [8] [LQR09], as a starting point for our implementation. It is a symbolic model checker able to verify MAS against temporal-epistemic specifications with cooperation modalities. Properties are specified in the logic ATLK, a suitable extension of ATL with epistemic operators and cooperation modalities [LQR09]. Observe that being CTL subsumed by ATL, and being ATLK an extension of ATL, it turns out that MCMAS can also be employed as a classical model checker for reactive systems. This suggests that while the approaches are very similar in spirit, MAS verification is, in fact, a (non-trivial) extension of reactive system verification (observe also that ATL* subsumes full CTL* [AHK02]). Of course, this raises new challenges, as the epistemic modalities require, in general, a different treatment. Details about the use of MCMAS as a verification tool in the ACSI project are provided in D2.2.1.

We remark that, in general, verification techniques for finite-state systems may represent a viable option for the verification of infinite-state systems. Indeed, one can provide a finite, high-level description of an infinite-state system, by taking into account only a finite number of (propositional) properties of interest. In other words, one can construct a coarse-grained system model and execute the verification procedure on it. Of course, this approach does not guarantee all specifications of interest to be verifiable, but may provide answers in some cases of interest. This approach is the basic idea behind *abstraction*, i.e., the process of simplifying a given model by only considering the aspects that are relevant to the particular property to be verified. Existing approaches to abstraction in verification are addressed later on in this section.

3.1.2 Infinite-state systems

The main problem associated with the verification of infinite-state systems is the general impossibility of performing an exhaustive search in the state space. Classical algorithms for model checking finite-state systems essentially fail because there is no guarantee that a fixpoint is eventually reached after a finite number of iterations. In fact, it is easy to find examples of undecidable verification tasks for infinite-state systems (e.g., reachability of a halting state in a Turing Machine).

Infinite-state systems often occur in the practice, and their verification is certainly of great practical interest. For instance, any C or Java program is potentially infinite-state, and so do artifact-based systems, as artifacts may contain values from infinite domains.

In the last two decades, the scientific community has paid great attention to this problem. System models have been proposed to capture relevant properties of infinite-systems, together with appropriate formalisms for requirement specification, classes of decidable (and undecidable) problems have been identified, and corresponding algorithms have been developed.

[8] http://www-lai.doc.ic.ac.uk/mcmas/

Technically, the definition of the model checking problem for the infinite-state case is exactly the same as for the finite-state case, that is, given a transition system and a property over it, check whether the property holds in the transition system. However, as said above, the infiniteness of the transition system makes the algorithms for classical model checking essentially useless in the general case.

A number of solutions have been proposed to deal with state infiniteness.[9] Many of them are based on identifying interesting classes of transition systems, definable by suitable formalisms, and some respective classes of decidable properties. Two fundamental results in this respect are [Büc62] and [GS84], where properties represented in Monadic Second-order Logic (MSO) are shown decidable respectively over linear orders and complete binary trees (representing the transition systems structure).

From these, a series of more general results are derived, over more complex structures obtained by suitable manipulations of basic transition systems, for which decidability results are known. Examples of such manipulations include *transductions* [Cou94], *tree-iterations* [Wal02] and *unfoldings*, that can be intuitively thought of as operations that build transition systems out of transition systems, by preserving some regularities that can be exploited by the verification algorithms.

For instance, in [Cau02], starting from the infinite complete binary tree, a hierarchy of infinite transition systems whose MSO-theories (i.e., sets of MSO properties defined over them) are decidable is obtained by means of *MSO-definable interpretations* [Cou94] and *unfoldings*, these representing operations similar to those described above. Similarly, other results state that the tree-iteration transformation preserves decidability of MSO theories [Sem84, Wal02]. On the same line, decidability of MSO theories over transition systems obtained from pushdown automata or prefix-rewriting systems is also proven [Cau03, MS85]. Concerning pushdown systems, it is also worth mentioning the existence of symbolic techniques used for reachability analysis [BEM97], and model checking algorithms for CTL properties [Wal00].

Another approach is *regular model checking* [BJNT00, JN00, BHV04], a uniform paradigm for algorithmic verification of several classes of parameterized and infinite-state systems. In this account system states are captured by strings of arbitrary length over a finite alphabet, and the transition relation is given by a regular, length-preserving relation on strings, usually represented by a finite-state transducer. The fundamental problem of computing the set of reachable states from a given initial configuration, or *reachability analysis*, is tackled by using two complementary techniques: an automata-theoretic construction, and a fixpoint computation. Differently from the approaches discussed above, regular model checking is in general incomplete. Observe that, although sufficient conditions on the transition relation that guarantee termination are identified [BJNT00], incomplete techniques are useful in the practice even without them. Indeed, an incomplete algorithm can be executed over a finite temporal horizon, and provide useful information in case of termination (while not allowing for concluding anything, in general, when it does not terminate).

Other proposals and studies on model checking of infinite-state systems do exist –such as [BG04], [FL02] and [KMM$^+$01], just to mention some, which tackle the problem from both the theoretical and the practical perspectives, and which are certainly relevant to ACSI as, ultimately, artifact-based systems can be seen as a particular class of infinite-state systems. However, a distinctive feature of artifact-based systems is the presence, in each state, of data with an explicit *relational* structure, and its relevance to both the system evolution and the properties

[9] Part of the following discussion follows [MP06], where many results in the area are reported and summarised.

of interest. In particular, differently from previous proposals for infinite-state model checking, states are interpreted over relational structures (over infinite domains) –and their properties are typically captured by quantified languages such as first-order logic, and the specifications of interest express properties concerning the evolution of such structures. This ultimately requires a specification language that combines (temporal and/or epistemic) modalities with quantifiers, thus allowing for comparing the data content and the relational structure of one state with that of other state(s). The existing verification approaches that deal with infinite data typically focus on the data values assigned to state variables, but do not allow for expressing nor verifying relational properties of single states or involving different states.

Therefore, while, on the one hand, the existing approaches and techniques offer a solid background for the development of actual verification procedures, on the other hand, they are not applicable *as they are*, and additional theoretical tools are needed in order to achieve a complete understanding of how data with an explicit structure represented in system states should be treated. While these points will be discussed more in details later on, here we review the most relevant work on this topic.

One of the earliest work on this topic is [AVFY00], where *relational transducers* are proposed as a computational model of e-commerce applications. These are abstract machines whose states are instances of a relational database, and that can interact with external actors by taking in input and possibly returning in output some relations. The evolution of the system, as well as its output, depends on the current state of the system (i.e., the current state of the database) and the input it receives. The relation transition is specified in a declarative, relational language (a restriction of FO), which essentially states the relationship among the current state, the input, the next state, and the output. Several problems of interest are defined on such model, the most relevant to ACSI probably being verification of temporal properties involving data evolution. Importantly, the expressiveness of the language adopted has a significant impact on the decidability of the verification tasks, so several restrictions are imposed. In particular, it is required that the state is able only to accumulate all inputs received up to the current state. While this clearly limits the expressive power of a relational transducer, called, under the above restrictions, *Spocus transducer*, the model is still able to enforce non-trivial temporal properties on the system's (finite) runs.

The Spocus transducer of [AVFY00] is generalised into the *Abstract state machine (ASM)* relational transducer in [Spi03]. In this new model, the limitations on the state evolution are weakened, by essentially allowing more general FO formulas to occur in the specifications of the transition relation, while preserving decidability of verifying a temporal formula (over infinite runs) built using first-order formulas as atomic blocks.

ASM transducers are, in turn, extended in [DSVZ06, DSV07], where an enriched model, called ASM^+ transducer, is introduced. The relevance of ASM^+ transducers essentially resides in their ability to capture *data-driven* Web applications and to enable verification of FO temporal properties, like those considered in [Spi03], under analogous restrictions. Such restrictions constrain the way that elements in the current state can be accessed, by essentially introducing a form of guarded quantification on both the specification of the transition relation and the property to verify.

All the above transducers are, in general, infinite-state, as the databases (although finite) can take values from an infinite interpretation domain, thus giving raise to an infinite number of (finite) instances, whereby the difficulty of the verification task. Importantly, the (constructive) decidability proof developed in [DSV07] shows how, by means of an abstraction process made possible by the guarded quantification restriction, the verification problem for an (infinite-state) ASM^+ transducer can be reduced to classical model checking over a finite structure, built starting

from the original transducer and the property to verify. Notably, this reduction represents the core of a verification procedure actually implemented in a real system [DMS+05].

The relevance of these works to the ACSI project becomes even more apparent in recent work [DHPV09] where a variant of the ASM^+ framework presented above is adapted and extended to capture so-called *data-centric business processes*, which essentially correspond to particular classes of artifact systems. Furthermore, recently, in [DDV11] a variant of the framework adopted in [DHPV09] is proposed, where the verification task has been proven decidable for (suitable classes of) artifact systems in presence of data dependencies and arithmetic operations.

As it turns out, while there exist a large pool of results about infinite-state systems where data is not explicitly considered, much (theoretical and practical) work is still needed to verify systems that store and exchange data with the environment, as it is the case for artifact systems. Moreover, the work on the former focussed, so far, on finding classes of systems and problems for which the verification task is decidable, and in particular allowing for a reduction to finite-state model checking, while no incomplete techniques of practical relevance have been investigated yet, to the best of our knowledge. Finally, we remark the great relevance of exploring the connections between verification techniques that take data into account and those that do not, as it seems reasonable to expect the latter providing useful guidelines for the former.

3.2 Synthesis

Intuitively, *synthesis* is the problem of building a program that satisfies a given specification. Originally known as *Church's problem* [Chu62], in the last two decades the community of formal verification has paid great attention to this problem. Its importance becomes clear by observing that while verification allows for discovering flaws in a system *after* it has been built, synthesis allows for obtaining a system that is correct by construction, hence yielding a dramatical reduction of the effort needed to guarantee compliance of the system with the intended behavior.

Open systems are central to synthesis. These are finite-state systems able to interact with the environment they act in, by reading and changing the (finite) values of a (finite) set of variables, over time. Such a model turns out to be quite expressive. For instance, it is able to model the evolution of a protocol that responds to incoming messages, or the behavior of a lift that, upon receiving a request, changes its position accordingly, and becomes ready to serve new requests. An open system essentially captures the way a system reacts to the stimuli incoming from an external environment. Precisely, by *synthesis*, we refer to the construction of a *controller* module (typically finite-state‘) that, when interacting with the system (e.g., the protocol or the lift) in a *closed loop*, results in a behavior guaranteed to satisfy a desired temporal specification. Intuitively, we can think of the system as a car, the controller as the driver, and the desired specification as, e.g., “drive safely around a building”.

Cornerstones in this area are [BL69] and [Rab72], where general solutions to Church's problem have been proposed, and, more directly related to this project, [PR89] where a procedure for synthesizing a controller (a.k.a. *reactive module*) starting from an LTL specification is provided. The obtained algorithm is based on checking emptiness of Rabin automata on infinite trees, and has a doubly exponential bound in the length of the specification. While representing a notable achievement from a theoretical point of view, the approach, due to its prohibitive complexity, turns out to be ineffective in practice.

Importantly, [PR89] characterises synthesis as a form of verification on *game structures*, which are essentially transition systems along whose runs two types of states alternate: those

representing the (open)system's actions, and those representing the environment's replies. The synthesis task then amounts to guaranteeing that the system has a strategy to fulfill the specification, no matter how the environment replies to its actions (along every run). The tight relation between synthesis and verification becomes even more apparent in the case of MAS verification where, e.g., verification of ATL formulas reduces, in fact, to a particular class of synthesis, in that the existence of a strategy to satisfy a specification is checked for (although the strategy itself is not computed).

To overcome the obstacle represented by the high problem complexity, and make synthesis for LTL specifications a task practically feasible, some efforts have been made towards identifying classes of more tractable specifications of practical interest [PPS06, AMPS98, AT04]. In particular, [PPS06], whose results can be seen as a generalisation of those in [AMPS98, AT04], presents an actual algorithm for the synthesis of so-called *generalized reactivity (1)* formulas (GR(1)), for which the synthesis task is singly exponential, and provides examples showing the practical interest of such class of specifications. This algorithm has been implemented (together with others) in the system TLV [10] (temporal logic verifier) and, recently, in its new, Java-based extension JTLV [PSZ10].

It is worth noticing that the above results served as basis for a series of works on the composition of behavioural services and agents [GMP09, GP09, GFPS10], where it is shown how the problem of building a desired behavior by suitably coordinating, through a synthesized controller, a set of given behaviors, can be reduced to an instance of the LTL synthesis problem. Based on the interpretation of the stakeholders as agents interacting with the artifact system, we believe that these results can provide a solid bases in tackling synthesis problems that involve the interaction between stakeholders and artifact system.

Finally, we mention [BCG+05], which is the only result we are aware of about (a form of) synthesis in presence of data. This work presents a technique for constructing a *mediator* that, placed in the middle between a set of services (able to exchange messages) and a set of users, suitably coordinates the services so as to provide a new, desired service, not available otherwise. Similarly to those discussed above, this work, though developed in a different setting, could provide useful insights on the development of techniques for the synthesis of control procedures that guarantee stakeholders to interact with the system, while satisfying desired requirements (e.g., access restrictions to particular artifacts).

3.3 Abstraction

Generally speaking, *abstraction* in verification is a technique for simplifying finite- or infinite-state systems by removing details that are not relevant to the property under verification. Typically, abstraction techniques amount to *clustering* states of the original system that satisfy some common properties into so-called *abstract states*, then deriving abstract transitions among such abstract states, and finally verifying the obtained (abstract) system. Different criteria defining how states are clustered and how the transition relation is derived, give raise to different abstraction approaches. However, all of them have in common the fact that since the abstract state space is typically smaller than the concrete space, the approach generally results in a significant reduction on the execution time of the verification algorithm, or in the possibility of verifying a property otherwise unverifiable, in the case of infinite-state systems. Of course, this advantages do not come, in general, for free, but require some limitations on the properties that

[10] `www.cs.nyu.edu/acsys/tlv/`

can be verified on the abstract system.

Although abstraction as a mean for program verification dates back to the late 70's, with the foundational work by P. Cousot and R. Cousot [CC77], who proposed a general framework for the construction of sound approximations (abstractions) of mathematical structures, here we review more recent results, in particular those related to model checking, as directly relevant to the ACSI project.

One of the most popular abstraction approaches in model checking is described in [CGJ+03] (that can be seen as an extension of the earlier *existential abstraction* framework presented in [CGL94]), where a technique, known as *counter-example guided abstraction refinement* (CEGAR), is presented. It is proven complete over the class of CTL* formulas that admit a single run as counterexample (namely, ACTL*), if the original system is finite-state.

The proposed algorithm intuitively works as follows:

1. an initial abstraction is derived from a compact specification (program) of the concrete model and the property under verification. In general, this step produces an abstract model that *over-approximates* the original model, that is, it contains *spurious* transitions not capturing any concrete transition;

2. the current abstract model is verified against the property of interest;

3. if the property is satisfied in the abstract model, it is guaranteed to be satisfied in the concrete model, too, and the process terminates successfully;

4. otherwise, the counterexample returned by the verification step is checked against the concrete model, to see if it captures any concrete behavior;

5. if it does, the process terminates, with the property of interest guaranteed not being satisfied by the concrete model;

6. if it does not, the current abstract model is refined by removing the spurious transitions, and a new iteration starts from step 2.

Even though the approach does not guarantee completeness for infinite-state systems, it still provides a viable, though incomplete, option in these cases. For instance, one can fix a bound on the number of iterations on the algorithm, and try to check the property of interest over the abstract models within the fixed bound. If the process terminates before reaching the bound, then the answer it provides is correct, while nothing can be said otherwise.

As discussed above (see Section 2), artifact system verification has many similarities with classical verification (of finite-state systems), in that the former can be seen as transition systems whose states have a particular (data) structure. Also, from our discussion above, it turned out that temporal-epistemic properties with first-order features, that can be seen as generalising classical temporal properties in model checking, are of interest for artifact systems. Moreover, the artifact-centric paradigm includes a formal, compact, representation of artifact systems, i.e., GSM or FSM, as discussed in WP1 report for M1.

These observations candidate the verification approach of [CGJ+03] as a possible starting point for the development of effective verification procedures for artifact systems. In particular, if an abstraction procedure for artifact systems were available that would guarantee, similarly to the propositional case, the preservation of interesting properties from the abstract to the concrete model, one could adapt the general structure of the CEGAR algorithm, so as to obtain an effective, though possibly incomplete, verification procedure for artifact systems.

However, two main obstacles make such an extension challenging: firstly, the presence of data, that makes the system infinite-state, and the relevance of its organization to the properties of interest, and, secondly, the interest in epistemic properties, that adds a further dimension to take into account in the verification procedure. As for the latter issue, efforts in that direction have already been undertaken. In particular, [CDLR09] adapts the existential abstraction approach proposed in [CGL94] to the context of *interpreted systems*, a well-known framework constituting the theoretical bases of MAS's and their corresponding (propositional) epistemic logics [FHMV95]. In Section 4, we further extend this work to the framework of artifact-systems, thus taking a first step towards the development of abstraction-based verification procedures in the artifact context. Such an extension finds its theoretical bases in recent work [BL09], where the standard (propositional) framework of interpreted systems, used to model multi-agent systems and to reason about agents knowledge, is extended to account for generic first-order properties. This results in a more general model that is particularly well-suited also for artifact systems (see Section 4 for details).

The abstraction approach of [CGJ$^+$03] is not the only one proposed in the model checking context. Another similar approach, known as *predicate* (or *boolean*) abstraction, based on partitioning the state space of the concrete system according to relevant properties satisfied by each state is proposed in [GS97] and [Saï00]. Despite the approach being slightly more general, it achieves analogous results, in particular the preservation of ACTL* satisfaction from the abstract to the concrete level, the possibility of iteratively refining the abstract model by exploiting the obtained counterexamples, and the incompleteness result for infinite systems. Therefore, from the ACSI point of view, the challenges to be faced in order to suitably extend these approaches happen to be exactly the same.

The abstraction approaches described so far are tailored on propositional model checking. While their extension to the ACSI framework is certainly worth exploring, this is not the only available option to build effective verification procedures for artifact systems. Of course, (complementary) *ad-hoc* procedures tailored on artifact systems can also be developed. With respect to this, we mention again the series of works [DSVZ06, DSV07, DHPV09, DDV11], where abstraction techniques well-suited for systems that deal with explicit data are developed, that we believe can be adapted in the context of ACSI, and possibly extended to take the epistemic properties into account.

In Section 5, we describe a framework for the verification of temporal properties, involving data, over artifact systems. In such a work, an ad-hoc abstraction technique is developed to deal with new values that actor interacting with the system can possibly introduce as the system evolves.

3.4 Discussion

In this section we have reviewed the literature on verification and synthesis of finite- and infinite-state systems, that is most relevant to the ACSI project, and in particular to Task 2.1. This includes classical approaches for finite-state systems, such as model checking and synthesis for temporal properties, for infinite-state systems, and *ad-hoc* approaches to deal with data.

Since artifact systems can be seen as transition systems, we expect model checking approaches, both in the finite- and the infinite-state case, to provide useful insights about verification of artifact systems, even including the possibility of extending existing algorithms to deal with artifact systems.

However, we have identified two major obstacles that need to be overcome before MC-based techniques can become effective over artifact systems. Precisely, such obstacles arise from the presence of infinite data (and the fact that properties referring to the actual data content of the system are of interest), and the fact that such properties can be, in general, epistemic. Since verification of systems dealing with data is a little explored area, and even less verification of epistemic properties in presence of data, a major effort on the theoretical side is certainly required in order to understand how the problem can be effectively addressed.

Previous works of particular interest, which can serve as starting points in our investigation include [DHPV09, DSV07] and [CDLR09, BL09]. The former provide some bases for reducing MC over infinite systems that deal with data to MC over finite systems, while the latter both extend a popular abstraction technique to the framework of interpreted systems and introduce an extension of interpreted systems able to capture data in the state of the system. We believe that ideas from these works can be successfully combined so as to build a new framework and actual procedures for the verification of temporal and temporal-epistemic properties over artifact systems.

Finally, we observe that existing approaches are typically aimed at identifying decidable classes of systems and/or properties that guarantee decidability. Considering that such decidable classes yield an (exponential) complexity bound that makes the problem, in general, unfeasible, it turns out that also incomplete techniques, possibly used in conjunction with complete ones, represent a further viable option to address the problem, certainly worth exploring.

4 A Computationally-Grounded Semantics for Artifact-Centric Systems

In this section we provide a first foundational stepping stone towards verification of temporal-epistemic properties of artifact-centric systems. Inspired by well-known results in logic and knowledge representation [FHMV95], we put forward a formalisation of artifact systems by providing them with a computationally-grounded semantics [Woo00] that we illustrate on a concrete scenario. The semantics is used to interpret a first-order language that includes temporal and epistemic modalities to model the temporal evolution of the system's properties including the information the agents hold. We address the model checking problem, which, besides being interesting *per se*, provides a basis for solutions of many typical problems in this area, including orchestration and choreography. This is a difficult problem as both the quantification domain and the state-space are infinite. We make an initial, yet promising dent into this by giving abstraction results that, in some cases, permit the reduction of the problem to model checking on finite domains.

4.1 Formal Model

We recall the notion of database and then introduce *artifact quantified interpreted systems* (A-QIS), which we show to be well-suited to provide a semantics to Artifact-Centric systems (ACS).

Definition 1 (Database Schema)*:* A *database schema* is a set $D = \{R_1, \ldots, R_n\}$, where each R_i is a *relation schema* of the form $R_i = r_i(a_1, \ldots, a_{k_i})$, and all a_js are pairwise distinct. For each

relation schema R_i we refer to r_i as the *relation* name, a_j as the j-th *relation attribute*, and k_i as R_i's (or r_i's) *arity*.

Definition 2 (Database Instance)*:* An *instance* (or *interpretation*) D of a database schema D over a possibly infinite *interpretation domain* V is a set of finite relations $\mathsf{D} = \{\mathsf{R_1}, \ldots, \mathsf{R_n}\}$ such that $\mathsf{R}_i \subseteq V^{k_i}$, for $i = 1, \ldots, n$. Each R_i is called *instance* (or *interpretation*) of (relation schema) R_i.

For a schema D, write $\mathcal{I}_D(V)$ (or simply $\mathcal{I}_D$, if V is clear from the context) for the set of all D's interpretations over V.

We capture ACSs by using ideas from Intepreted Systems semantics [FHMV95] and extensions [BL09].

We assume a set $Ag = \{1, \ldots, m\}$ of agents, an environment e, a database schema $D = \{R_1, \ldots, R_n\}$, and an alphabet $\mathcal{A}$ containing individual constants $c_1, c_2, \ldots$, n-ary predicate letters $P_1^n, P_2^n, \ldots$ for $n \in \mathbb{N}$, as well as all relation schemes $R_1, \ldots, R_n$ in D. Further, for each agent $i \in Ag$ we introduce a set L_i of local states $l_i, l_i', \ldots$, a set ACT_i of actions $\alpha_i, \alpha_i', \ldots$, and a protocol function $p_i : L_i \mapsto 2^{Act_i}$. We consider local states, actions and a protocol function for the environment e as well. The set $\mathcal{S} \subseteq L_1 \times \ldots \times L_m \times L_e$ contains the global states of the systems; while $Act \subseteq Act_1 \times \cdots \times Act_m \times Act_e$ and $p = \langle p_1, \ldots, p_m, p_e \rangle$ are the set of joint actions and the joint protocol respectively.

In order to introduce artifact quantified interpreted systems (A-QIS), we define a *view on a database instance* as the set of relations returned by a set of queries (precisely, one relation per query) on the database instance, and a query, as standard, as a mapping from the set of database instances to the set of relations (of given arity) over the interpretation domain.

Definition 3 (A-QIS)*:* An *artifact quantified interpreted system* is a tuple $\mathcal{P} = \langle D, V, s_0, \tau, I \rangle$ where:

- for each $i \in Ag$, D_i is a view (see below) on D, and $L_i = \mathcal{I}_{D_i}(V)$. Also, $L_e = \mathcal{I}_D(V)$;
- V is the interpretation domain of D;
- $s_0 \in \mathcal{S}$ is the initial global state;
- $\tau : \mathcal{S} \mapsto (Act \mapsto \mathcal{S})$ is the transition function, where $\tau(s)(\alpha)$ is defined only if $\alpha \in p(s)$.
- I is an interpretation of the alphabet $\mathcal{A}$ such that: *(i)* for every constant $c \in \mathcal{A}$, $I(c) \in V$; and *(ii)* for every predicate letter $P^n \in \mathcal{A}$, $I(P^n, s) \subseteq V^n$. In particular, for each relation schema R, $I(R, s) = \mathsf{R}$.

For $s, s' \in \mathcal{S}$ we say that s' is a *successor* of s, or $s \to s'$, if there exists $\alpha \in Act$ such that $s' = \tau(s)(\alpha)$. A *run* r is a sequence $r = s^0 \to s^1 \to \ldots$ such that $s^i \to s^{i+1}$. For $n \in \mathbb{N}$, $r(n)$ is the n-th element in the sequence, i.e., s^n. For $s, s' \in \mathcal{S}$ we say that s is *epistemically indistinguishable* from s' for agent i, or $s \sim_i s'$, if $s_i = s_i'$ [FHMV95].

Observe that A-QISs specialise quantified interpreted systems (QIS), as defined in [BL09]. In fact, a QIS can be seen as an A-QIS $\mathcal{P} = \langle V, s_0, \tau, I \rangle$ with no database schema, so that the internal structure of local states is left unspecified.

4.2 The First-Order MAS Logic FO-CTLK

Given the set Ag of agents and the alphabet $\mathcal{A}$, the first-order temporal epistemic language $\mathcal{L}_m$ contains all individual constants, predicate letters and relation schema in $\mathcal{A}$, individual variables

$x_1, x_2, \ldots$, the connectives $\neg$ and $\rightarrow$, the quantifier $\forall$, the branching time operators AX, AU and EU, the epistemic operator K_i for each agent $i \in Ag$, and the common knowledge operator C [FHMV95]. $\mathcal{L}_m$ contains no functional symbols, so the only terms $t_1, t_2, \ldots$ in the language are individual variables and constants.

Definition 4*:* Formulas in $\mathcal{L}_m$ are defined in BNF as follows:

$$\phi ::= P^k(t_1, \ldots, t_k) \mid \neg\phi \mid \phi \rightarrow \phi \mid \forall x\phi \mid AX\phi \mid A\phi U\phi \mid E\phi U\phi \mid K_i\phi \mid C\phi \tag{1}$$

The formulas $AX\phi$ and $A\phi U\phi'$ (resp. $E\phi U\phi'$) are read as "*for all paths, at the next step* ϕ" and "*for all paths (resp. for some path),* ϕ *until* ϕ'". $K_i\phi$ means "*agent* i *knows* ϕ"; while $C\phi$ is read as "ϕ *is common knowledge*". The logical symbols $\wedge$, $\vee$, $\leftrightarrow$ and $\exists$, as well as the operators AG, AF, EG, EF, and EX are defined as standard. Finally, $E\phi$ is defined as $\bigwedge_{i \in Ag} K_i\phi$, and for $n \in \mathbb{N}$, $E^0\phi = \phi$ and $E^{n+1}\phi = EE^n\phi$.

By $\phi[\vec{y}]$ we mean that $\vec{y} = y_1, \ldots, y_n$ are all ϕ's free variables; and $\phi[\vec{y}/\vec{t}]$ is the formula obtained by substituting simultaneously some, possibly all, free occurrences of $\vec{y}$ in ϕ with $\vec{t} = t_1, \ldots, t_n$ while renaming bound variables. As standard, a sentence is a formula with no free variables.

In what follows we consider two fragments of $\mathcal{L}_m$.

Definition 5*:* Formulas in the $\forall$ACTLK-fragment of $\mathcal{L}_m$ are defined in BNF as follows, where the logical symbols $\vee$, $\wedge$, $A\overline{U}$, and $\exists$ are taken as primitives:

$$\phi ::= P^k(\vec{t}) \mid \neg P^k(\vec{t}) \mid \phi \vee \phi \mid \phi \wedge \phi \mid \forall x\phi \mid AX\phi \mid A\phi U\phi \mid A\phi\overline{U}\phi \mid K_i\phi \mid C\phi \tag{2}$$

where $A\phi\overline{U}\phi'$ is read as "*for all paths,* ϕ *release* ϕ'".

The $ACTLK$-fragment extends the $\forall ACTLK$-fragment with the following clause:

- if ϕ is a formula, then $\exists x\phi$ is also a formula.

We now define the semantics of $\mathcal{L}_m$-formulas in terms of A-QISs. Given an assignment σ from the set of variables in $\mathcal{L}_m$ to the individuals in V, the interpretation $I^\sigma(t)$ of an individual term t is defined as $\sigma(t)$ if t is a variable, or $I(t)$ if t is a constant. Also, σ^x_a is the assignment that maps x to a and coincides with σ on all other variables.

Definition 6*:* The satisfaction relation $\models$ for $\phi \in \mathcal{L}_m$, $s \in \mathcal{S}$, and an assignment σ is inductively defined as follows:

$(\mathcal{P}^\sigma, s) \models P^k(\vec{t})$ iff $\langle I^\sigma(t_1), \ldots, I^\sigma(t_k)\rangle \in I(P^k, s)$
$(\mathcal{P}^\sigma, s) \models \neg\phi$ iff $(\mathcal{P}^\sigma, s) \not\models \phi$
$(\mathcal{P}^\sigma, s) \models \phi \rightarrow \phi'$ iff $(\mathcal{P}^\sigma, s) \not\models \phi$ or $(\mathcal{P}^\sigma, s) \models \phi$
$(\mathcal{P}^\sigma, s) \models \forall x\phi$ iff for all $a \in V$, $(\mathcal{P}^{\sigma^x_a}, s) \models \phi$
$(\mathcal{P}^\sigma, s) \models AX\phi$ iff for all runs r, if $r(n) = s$ then $(\mathcal{P}^\sigma, r(n+1)) \models \phi$
$(\mathcal{P}^\sigma, s) \models A\phi U\phi'$ iff for all runs r, if $r(n) = s$ then
there is $k \geq n$ s.t. $(\mathcal{P}^\sigma, r(k)) \models \phi'$, and $n \leq k' < k$ implies $(\mathcal{P}^\sigma, r(k')) \models \phi$
$(\mathcal{P}^\sigma, s) \models E\phi U\phi'$ iff for some run r, $r(n) = s$, and
there is $k \geq n$ s.t. $(\mathcal{P}^\sigma, r(k)) \models \phi'$, and $n \leq k' < k$ implies $(\mathcal{P}^\sigma, r(k')) \models \phi$
$(\mathcal{P}^\sigma, s) \models K_i\phi$ iff for all s, s', if $s \sim_i s'$ then $(\mathcal{P}^\sigma, s') \models \phi$
$(\mathcal{P}^\sigma, s) \models C\phi$ iff for all $k \in \mathbb{N}$, $(\mathcal{P}^\sigma, s) \models E^k\phi$

A formula $\phi \in \mathcal{L}_m$ is *true* in a state s, or $(\mathcal{P}, s) \models \phi$, if for all assignment σ, $(\mathcal{P}^\sigma, s) \models \phi$; it is *true* in an A-QIS $\mathcal{P}$, or $\mathcal{P} \models \phi$, if $(\mathcal{P}, s_0) \models \phi$.

4.3 The Equipment Purchasing Scenario

We now introduce a scenario, inspired by a real Business Process use-case identified by IBM, which shows the artifact-centric approach at work. Although we do not formally define a general translation procedure from artifact-centric business processes to A-QIS, this formalisation exemplifies how this can be done and, therefore, demonstrates that A-QISs can effectively be employed to reason about real ACSs.

4.3.1 The Scenario

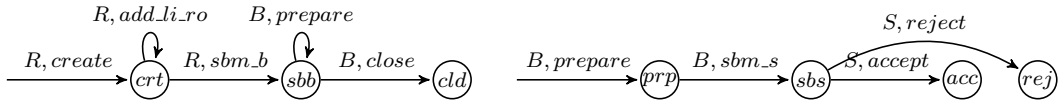
The employees of a company who need to purchase some equipment must follow a procedure, which involves the following agents: a *requester*, who needs to purchase the equipments; a *buyer*, who is in charge of buying the requested items; and some *suppliers*, who supply the equipment. The process begins when the requester fills out a *requisition order* with some line items referring to desired products (e.g., a laptop). The requester then submits the order to the buyer, who finds an appropriate supplier for each line item, and prepares a *procurement order* containing the relevant line items. Procurement orders are then submitted to suppliers, who can either reject or fulfill the received orders. In the former case, the buyer is notified; in the latter, the items are shipped.

Orders are captured by *artifacts*, containing attributes, and associated with sets of actions enabling their manipulation. Requisition orders contain fields: *id*, the requisition order identifier; *itm*, the line item(s) occurring in the order; *p_ord*, the procurement order(s) associated with the requisition order; and *status* (see below). Procurement orders contain: *id*, the procurement order identifier; *s_id*, the identifier of the supplier selected by the buyer; *itm*, the line item(s) occurring in the order; and *status* (see below).

The evolution of an artifact *status* field, in response to agent actions, is referred to as the artifact's *lifecyle*, which accounts for the stage of the process that the artifact is in. Figures 1(a) and 1(b) depict the lifecycles for request and procurement orders in the form of transition systems, where nodes are labelled by the status they represent, and transitions by pairs "ag, op", associating each action (op) with the agent (ag) that can execute it, R standing for requester, B for buyer, and S for supplier. Each request order has 3 possible statuses, namely *crt* (created), *sbb* (submitted to buyer), and *cld* (closed), and 5 transitions: $R, create$ (the requester creates a request order); R, add_li_ro (the requester adds a line item to the created request order); R, sbm_b (the requester submits the order to the buyer); $B, prepare$ (the requester prepares a procurement order for this request order); $B, close$ (the requester closes the request order, when all items are shipped). Similarly, procurement orders have 4 possible statuses, i.e., *prp* (prepared), *sbs* (submitted to supplier), *rej* (rejected), *acc* (accepted), and 4 transitions: $B, prepare$ (the buyer prepares the procurement order); B, sbm_s (the buyer submits the procurement order to the supplier); $S, accept$ (the supplier accepts the order); $S, reject$ (the supplier rejects the order).

Each field of an artifact can, in general, be affected by any action, even if occurring in the lifecyle of other artifacts. For instance, *prepare* can be executed by the buyer when a requisition order *ro* is in status *sbb*, but has the effect of creating a new procurement order associated with *ro*. Also, there can be actions affecting an artifact's field but not the status, as it is the case, e.g., of *add_li_ro*, which adds a line item to a request order but does not change the order status.

Typically, agents are granted access only to the information they require for their tasks, an aspect taken into account in our formalisation. For instance, the requester is not informed about the suppliers selected to supply each item.



(a) Request order lifecyle (b) Procurement order lifecyle

Figure 1 – Lifecycles of request and procurement orders

4.3.2 Formalisation

For simplicity we assume each requisition order contains at most one item (and, consequently, is associated with at most one procurement order). However, A-QISs are sufficiently expressive to overcome this limitation, as the relationship among a requisition order and many items can easily be captured, as standard in databases, by a relation containing the item attributes plus an attribute referencing the requisition order each item belongs to.

Agents. The set of agents is $Agt = \{1, 2, \ldots, n_S, e\}$, where: 1 represents the requester; 2 the buyer; $3, \ldots, n_S$ stand for the suppliers; and e represents the environment.

Interpretation Domain. We take $V = ID_A \cup ID_S \cup LI$ as the interpretation domain, where: ID_A is an infinite set of *artifact identifiers*, $ID_S = \{3, \ldots, n_S\}$ is the finite set of *supplier identifiers* (see below), and LI is the finite set of all line items, i.e., $LI = \{monitor, printer, phone\}$. All these sets are assumed disjoint. Moreover, we take the symbol $- \in V$ to represent a null value.

Environment Database Schema. We define the A-QIS environment database schema $D_e = \{RO, PO\}$, where the relations RO and PO are intended to record the information about all request and procurement orders. Their schemes are as follows: $RO(id, itm, p\_ord, status)$, where: id is the requisition order identifier, itm is the (single) line item occurring in the order, p_ord is the (single) procurement order associated with the requisition order, and $status$ is the current status of the order; $PO(id, s\_id, itm, status)$, where: id is the procurement order identifier, s_id is the identifier of the supplier selected by the buyer, itm is the (single) line item in the order, and $status$ is the current order status.

Local States. To model the requester's (i.e., agent 1) local-state space we take the set $\mathcal{I}_{D_1}(V)$ of interpretations of the database $D_1 = \{RO_1\}$, where RO_1's schema matches RO's. Similarly, buyer's (i.e., agent 2) local states are interpretations of the database $D_2 = \{RO_2, PO_2\}$ over V ($\mathcal{I}_{D_2}(V)$) where RO_2's and PO_2's schemes match RO's and PO's, respectively. As for agents $3, \ldots, n_S$, the suppliers, local states are interpretations of databases $D_i = \{PO_i\}$ over V, i.e., $\mathcal{I}_{D_i}(V)$, for $i = 3, \ldots, n_S$, where PO_i's schema is the same as PO's Finally, as set of environment local states we simply take $\mathcal{I}_{D_e}(V)$.

Actions. For conciseness, without loss of generality, we use parametric actions. A parametric action of the form $a(p_1, \ldots, p_q)$, where a is the action name and $p_1, \ldots, p_q$ are its parameters, represents a family of actions containing one *ground* action per distinct parameter assignment of values from V. As a convention, for an action parameter p, we denote the generic value assigned to p as p. Agents' action sets are defined as follows. For agent 1, we take $Act_1 = \{create, add\_li\_ro(id, li), sbm\_b(id)\}$, where: $create$ is meant to create a new request order (with unique identifier); $add\_li\_ro(id, li)$ is meant to add line item li to the request order identified by id; and $sbm\_b(id)$ is meant to submit the request order identified by id to the buyer. The action sets for the other agents are similar. For agent 2 (buyer), we have $Act_2 = \{prepare(id), close(id), sbm\_s(id)\}$, where: $prepare(id)$ creates a new procurement order (with unique identifier), related to request order id; $close(id)$ closes the request order identified

by id; and $sbm\_s(id)$ submits the procurement order identified by id to the respective supplier. Supplier action sets are $Act_i = \{accept(id), reject(id)\}$ $(i = 3, \ldots, n_S)$, where $accept(id)$ and $reject(id)$ have the intuitive semantics of accepting and rejecting, respectively, the procurement order identified by id (if any). As for the environment, we simply have $Act_e = \emptyset$.

Protocol Functions. Concerning protocols, since agent states are, as said above, database instances, the possibility of executing an action ultimately depends on the content of such instances. We report the requester, buyer and suppliers protocol functions. Since Act_e is empty, no protocol is defined for the environment.

Requester protocol $p_1 : \mathcal{I}_{D_1}(V) \mapsto 2^{Act_1}$ is as follows ($\mathsf{D}_1 = \{\mathsf{RO}_1\}$): $create \in p_1(\mathsf{D}_1)$ for all $\mathsf{D}_1 \in \mathcal{I}_{D_1}(V)$, i.e., action $create$ is always available to the requester, no matter what the current state of artifacts is; $add\_li\_ro(\mathsf{id}, \mathsf{li}) \in p_1(\mathsf{D}_1)$ iff $\mathsf{li} \in LI$ and there exists a tuple $\langle \mathsf{id}, \mathsf{crt} \rangle \in \mathsf{RO}_1$, i.e., the requester can add a list item only to request orders whose current status is crt; and $sbm\_b(\mathsf{id}) \in p_1(\mathsf{D}_1)$ if there exists a tuple $\mathsf{id}, \mathsf{itm}, \mathsf{p\_ord}, \mathsf{crt} \in \mathsf{RO}_1$, i.e., the requester can submit to the buyer only request orders whose current status is crt.

The buyer's protocol $p_2 : \mathcal{I}_{D_2}(V) \mapsto 2^{Act_2}$ is as follows ($\mathsf{D}_2 = \{\mathsf{RO}_2, \mathsf{PO}_2\}$). $prepare(\mathsf{id}, \mathsf{s\_id}) \in p_2(\mathsf{D}_2)$ if there exists a tuple $\langle \mathsf{id}, \mathsf{itm}, \mathsf{p\_ord}, \mathsf{sbb} \rangle \in \mathsf{RO}_2$, and either $\mathsf{p\_ord} = -$ or there exists a tuple $\langle \mathsf{p\_ord}, \mathsf{s\_id}, \mathsf{itm}, \mathsf{s} \rangle \in \mathsf{PO}_2$ s.t. $\mathsf{s} \in \{\mathsf{prp}, \mathsf{rej}\}$. This formalises that the buyer can prepare a new procurement order for an existing request order only if the request order is in status sbb, and either no corresponding procurement order has been prepared yet, or, if it has, it is either in status prp (prepared) or rej (rejected). $close(\mathsf{id}) \in p_2(\mathsf{D}_2)$ if there exists two tuples: $\langle \mathsf{id}, \mathsf{itm}, \mathsf{p\_ord}, \mathsf{sbb} \rangle \in \mathsf{RO}_2$ with $\mathsf{p\_ord} \neq -$, and $\langle \mathsf{p\_ord}, \mathsf{itm}, \mathsf{s\_id}, \mathsf{acc} \rangle \in \mathsf{PO}_2$, that is, the buyer can close only existing request orders whose associated procurement order is accepted; and $sbm\_s(\mathsf{id}) \in p_2(\mathsf{D}_2)$ if there exists a tuple $\langle \mathsf{id}, \mathsf{s\_id}, \mathsf{itm}, \mathsf{prp} \rangle \in \mathsf{PO}_2$, i.e., the buyer can submit (to a supplier) only prepared procurement orders.

Supplier protocols $p_i : \mathcal{I}_{D_i}(V) \mapsto 2^{Act_i}$ $(i = 3, \ldots, n_S)$ are such that ($\mathsf{D}_i = \{\mathsf{PO}_i\}$): $accept(\mathsf{id}), reject(\mathsf{id}) \in p_i(\mathsf{D}_i)$ if there exists a tuple $\langle \mathsf{id}, \mathsf{s\_id}, \mathsf{itm}, \mathsf{sbs} \rangle \in \mathsf{PO}_i$ such that $\mathsf{id} = i \in ID_S$, i.e., supplier i can accept or reject only procurement orders submitted to herself.

Global States. As a consequence of the way local states are defined above, for global states we simply have $S \subseteq \mathcal{I}_{D_1}(V) \times \mathcal{I}_{D_2}(V) \times \mathcal{I}_{D_3}(V) \times \ldots \times \mathcal{I}_{D_{n_S}}(V) \times \mathcal{I}_{D_e}(V)$.

Interpretation Function. The interpretation function I is simply defined as the identity on the environment's local state, that is, for $s = \langle \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, \ldots, \mathsf{D}_{n_S}, \mathsf{D}_e \rangle \in S$, $I(s) = \mathsf{D}_\mathsf{e}$.

Initial State. As for the (global) initial state, we assume all relations are initially empty.

Global Transition Function. First, we define some dependencies between the agents' local states and the environment's. For a global state $s = \langle \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, \ldots, \mathsf{D}_{n_S}, \mathsf{D}_e \rangle$, with $\mathsf{D}_1 = \{\mathsf{RO}_1\}$, $\mathsf{D}_i = \{\mathsf{PO}_i\}$, for $i = 3, \ldots, n_S$, and $\mathsf{D}_e = \{\mathsf{RO}, \mathsf{PO}\}$, we have: $\mathsf{RO}_1 = \mathsf{RO}$, i.e., the requester can access the same information about request orders as the environment; $\mathsf{D}_2 = \mathsf{D}_e$, that is, buyer's local states contain the same information as the environment's; and for each $i = 3, \ldots, n_S$, $\langle \mathsf{id}, \mathsf{s\_id}, \mathsf{itm}, \mathsf{status} \rangle \in \mathsf{PO}_i$ iff $\mathsf{s\_id} = i$ and $\langle \mathsf{id}, \mathsf{s\_id}, \mathsf{itm}, \mathsf{status} \rangle \in \mathsf{PO}$, that is, each supplier can access the same information about procurement orders assigned to her as the environment. For simplicity, given a current global state s and a joint action a, we define the successor global state $s' = \tau(a)(s)$ by defining its environment component $\mathsf{D}'_e = \{\mathsf{RO}', \mathsf{PO}'\}$, as all other components are derivable through the above dependencies.

We now describe the transitions each agent triggers when executing some action. For the requester, we have:

- For $a = \langle create, -, -, \ldots, - \rangle$, D'_e in s' is such that $\mathsf{PO}'_e = \mathsf{PO}_e$, and $\mathsf{RO}'_e = \mathsf{RO}_e \cup \{\langle \mathsf{id}', -, -, crt \rangle\}$, with $\mathsf{id}' \in ID_A$ such that there is no other tuple $\langle \mathsf{id}, \mathsf{itm}, \mathsf{p\_ord}, \mathsf{status} \rangle \in \mathsf{RO}$ such that $\mathsf{id} = \mathsf{id}'$. Informally, when the requester creates a new order, a new request

order artifact is created, with a unique identifier.

- For $a = \langle add\_li\_ro(\mathsf{id}, \mathsf{li}), -, -, \ldots, - \rangle$, let $\bar{t} = \langle \mathsf{id}, \mathsf{itm}, \mathsf{p\_ord}, \mathsf{status} \rangle \in \mathsf{RO}_2$ (such a tuple exists, by definition of p_1 and the dependencies above). Then, D'_2 in s' is such that $\mathsf{PO}'_2 = \mathsf{PO}_2$ and $\mathsf{RO}'_2 = (\mathsf{RO}_2 \setminus \{\bar{t}\}) \cup \{\bar{t}'\}$, where $\bar{t}' = \langle \mathsf{id}, \mathsf{li}, \mathsf{p\_ord}, \mathsf{status} \rangle$. In words, when the requester adds a line item to a request order, the current line item of the selected (through id) order is overwritten (or added, if $\mathsf{itm} = -$).

- For $a = \langle sbm\_b(\mathsf{id}), -, -, \ldots, - \rangle$, let $\bar{t} = \langle \mathsf{id}, \mathsf{itm}, \mathsf{p\_ord}, crt \rangle \in \mathsf{RO}_2$ (such a tuple exists, see above). Then, D'_2 in s' is such that $\mathsf{PO}'_2 = \mathsf{PO}_2$ and $\mathsf{RO}'_2 = (\mathsf{RO}_2 \setminus \{\bar{t}\}) \cup \{\bar{t}'\}$, for $\bar{t}' = \langle \mathsf{id}, \mathsf{li}, \mathsf{p\_ord}, sbb \rangle$. That is, when the requester submits a request order to the buyer, the order changes its state to *sbb* (and the change is propagated to all local states).

Buyer's actions trigger the following transitions:

- For $a = \langle -, prepare(\mathsf{id}, \mathsf{s\_id}), -, \ldots, - \rangle$, let $\bar{t}_r = \langle \mathsf{id}, \mathsf{itm}, \mathsf{p\_ord}, sbb \rangle \in \mathsf{RO}_2$ (such a tuple exists, by p_2 definition and the above dependencies). We distinguish two cases: $\mathsf{p\_ord} = -$ and $\mathsf{p\_ord} \neq -$. In the former case, let $\bar{t}'_p = \langle \mathsf{p\_ord}', \mathsf{s\_id}, \mathsf{itm}, prp \rangle$, where $\mathsf{p\_ord}' \in ID_A$ is such that for all tuples $\langle \mathsf{p}, \mathsf{s}, \mathsf{i}, \mathsf{st} \rangle$ in PO_2, $\mathsf{p\_ord}' \neq \mathsf{p}$. Then, D'_2 in s' is such that $\mathsf{RO}'_2 = (\mathsf{RO}_2 \setminus \{\bar{t}_r\}) \cup \{\bar{t}'_r\}$, where $\bar{t}'_r = \langle \mathsf{id}, \mathsf{itm}, \mathsf{p\_ord}', sbb \rangle$, and $\mathsf{PO}'_2 = \mathsf{PO}_2 \cup \{\bar{t}'_p\}$. Intuitively, when the buyer prepares, for first time, a new procurement order associated with a request order identified by id, the procurement order is added to PO_2 (and to the local state of the corresponding supplier) and the request order is updated with the (new) identifier of the procurement order ($\mathsf{p\_ord}'$). Observe that the supplier associated with the procurement order is specified by $\mathsf{s\_id}$, while the line item itm is "extracted" from the request order. As for the latter case, i.e., $\mathsf{p\_ord} \neq -$, let $\bar{t}_p = \langle \mathsf{p\_ord}, \mathsf{s\_id}_\mathsf{p}, \mathsf{itm}, \mathsf{status} \rangle \in \mathsf{PO}_2$ (such a tuple exists, as above). Observe that, in general, $\mathsf{s\_id}_\mathsf{p} \neq \mathsf{s\_id}$. Then, D'_2 in s' is such that $\mathsf{RO}'_2 = \mathsf{RO}_2$ and $\mathsf{PO}'_2 = (\mathsf{PO}_2 \setminus \{t_p\}) \cup \{\mathsf{t}'_p\}$, for $\mathsf{t}'_p = \langle \mathsf{p\_ord}, \mathsf{s\_id}, \mathsf{itm}, prp \rangle$. That is, when the buyer modifies an existing procurement order for a request order, the supplier is updated(identifiers are preserved) and the status is set to *prp* (prepared).

- For $a = \langle -, close(\mathsf{id}), -, \ldots, - \rangle$, let $\bar{t}_r = \langle \mathsf{id}, \mathsf{itm}, \mathsf{p\_ord}, sbb \rangle \in \mathsf{RO}$ (such a tuple exists, by p_2 definition and the above dependencies). Then, D'_e in s' is such that $\mathsf{PO}'_e = \mathsf{PO}_e$ and $\mathsf{RO}'_e = (\mathsf{RO}_e \setminus \{\bar{t}_r\}) \cup \{\bar{t}'_r\}$, where $\bar{t}'_r = \langle \mathsf{id}, \mathsf{itm}, \mathsf{p\_ord}, cld \rangle$. That is, when the buyer closes a request order, the corresponding status changes to *cld* (closed).

- For $a = \langle -, sbm\_s(\mathsf{id}), -, \ldots, - \rangle$, let $\bar{t}_p = \langle \mathsf{id}, \mathsf{s\_}id, \mathsf{itm}, prp \rangle \in \mathsf{PO}_2$ (such a tuple exists, see above). Then, D'_2 in s' is such that $\mathsf{RO}'_2 = \mathsf{RO}_2$ and $\mathsf{PO}'_2 = (\mathsf{PO}_2 \setminus \{\bar{t}_p\}) \cup \{\bar{t}'_p\}$, where $\bar{t}'_p = \langle \mathsf{id}, \mathsf{s\_id}, \mathsf{itm}, sbs \rangle$. That is, when the buyer submits a procurement order to a supplier, the procurement order's status changes to *sbs* (submitted to supplier).

Finally, suppliers' actions trigger the following transitions:

- For $a = \langle -, -, \ldots, accept(\mathsf{id}), \ldots, - \rangle$, assuming the i-th supplier executes the action, let $\bar{t}_p = \langle \mathsf{id}, \mathsf{i}, \mathsf{itm}, sbs \rangle \in \mathsf{PO}_\mathsf{i}$ (such a tuple exists, by p_i definition and the above dependencies). Then D'_e in s' is such that $\mathsf{RO}'_e = \mathsf{RO}_e$ and $\mathsf{PO}'_e = (\mathsf{PO}_e \setminus \{\bar{t}_p\}) \cup \{\bar{t}'_p\}$, where $\bar{t}'_p = \langle \mathsf{id}, \mathsf{i}, \mathsf{itm}, acc \rangle$.

- Similarly, for $a = \langle -, -, \ldots, reject(\mathsf{id}), \ldots, - \rangle$, assuming the i-th supplier executes the action, let $\bar{t}_p = \langle \mathsf{id}, \mathsf{i}, \mathsf{itm}, sbs \rangle \in \mathsf{PO}_2$. Then, D'_2 in s' is such that $\mathsf{RO}'_2 = \mathsf{RO}_2$ and $\mathsf{PO}'_2 = (\mathsf{PO}_2 \setminus \{\bar{t}_p\}) \cup \{\bar{t}'_p\}$, for $\bar{t}'_p = \langle \mathsf{id}, \mathsf{i}, \mathsf{itm}, rej \rangle$. That is, when a supplier accepts a procurement order, the procurement order's status changes to *acc*.

Once the system has been modelled, it is possible to verify particular specifications on it. For instance, the system described above satisfies formula

$$\varphi_1 = AG(\forall id_r, itm, p\ RO_e(id_r, itm, p, \mathsf{cld}) \rightarrow \exists s\ K_2 PO_e(p, s, itm, \mathsf{acc})),$$

which states that a request order can be in state *closed* only if the buyer knows that the corresponding procurement order has been actually accepted by some supplier (this corresponds to the precondition of action *close* according to protocol p_2). This property intuitively corresponds to a specification that should indeed be satisfied in the scenario considered.

Also formula

$$\varphi_2 = \forall id\ AG(\exists s, itm\ PO_e(id, s, itm, \mathsf{prp}) \rightarrow AF(PO_e(id, s, itm, \mathsf{acc}) \vee PO_e(id, s, itm, \mathsf{rej})))$$

can be verified, which ensures that all procurement orders, once prepared, will be eventually accepted or rejected by the supplier they have been submitted to.

Of course, other temporal-epistemic specifications of interest can be similarly formalised.

4.4 Model checking and Abstraction

We would like to be able to verify automatically any formula φ in $\mathcal{L}$ on a given A-QIS, i.e., to give an effective methodology for answering the model checking query $\mathcal{P} \models \varphi$. The (considerable) difficulty resides in the fact that $\mathcal{P}$ is, in general, an infinite structure. To make inroads into this problem we give an abstraction technique for A-QIS by extending the results presented in [CDLR09] to the case of infinite models. We present the results obtained in their fullest generality by giving them on structures built on arbitrary sets, rather than specific database views (Def. 3). This corresponds to the general class of quantified interpreted systems (QIS) as defined in [BL09]. Given A-QIS are a subclass of QIS, all the results here proved also hold for A-QIS.

4.4.1 Simulation

The standard notion of simulation for reactive systems states that a system simulates another if every behaviour of the latter is a behaviour of the former [CGL94]. Since ACTL operators quantify over all behaviours (runs), any ACTL property that holds in the simulating system holds also in the simulated system. To extend this preservation property to ∀ACTLK, we require that any epistemic possibility in the simulated system is matched by an epistemic possibility in the simulating system. Similar conditions apply to individuals.

Definition 7 (Sublanguage)*:* Let $\mathcal{L}$ and $\mathcal{L}'$ be first-order temporal epistemic languages as in Def. 4, with alphabets $\mathcal{A}$ and $\mathcal{A}'$ respectively. $\mathcal{L}'$ is a sub-language of $\mathcal{L}$, or $\mathcal{L}' \subseteq \mathcal{L}$, if $\mathcal{A}' \subseteq \mathcal{A}$.

We now define the notion of simulation for QIS.

Definition 8 (Simulation)*:* Let $\mathcal{P} = \langle V, s_0, \tau, I\rangle$ be a QIS on the set Ag of agents and the language $\mathcal{L}$, and let $\mathcal{P}' = \langle V', s_0', \tau', I'\rangle$ be a QIS on the same set Ag of agents and a sub-language $\mathcal{L}' \subseteq \mathcal{L}$. A simulation between $\mathcal{P}$ and $\mathcal{P}'$ is a pair of relations $\simeq\ \subseteq \mathcal{S} \times \mathcal{S}'$ and $\approx\ \subseteq V \times V'$ such that:

(a) $s_0 \simeq s_0'$;

(b) if $a \in V$ then there exists $a' \in V'$ such that $a \approx a'$;

and if $s \simeq s'$ then:

(c) if $s \longrightarrow u$ then $s' \longrightarrow' u'$ for some u' such that $u \simeq u'$;

(d) if $s \sim_i u$ then $s' \sim'_i u'$ for some u' such that $u \simeq u'$;

(e) for all $P^n \in \mathcal{L}'$, if $\vec{a} \approx \vec{a}'$ then $\vec{a} \in I(P^n, s)$ iff $\vec{a}' \in I'(P^n, s')$;

(f) for all $c \in \mathcal{L}'$, $I(c) \approx I'(c)$;

where $\longrightarrow$ and $\longrightarrow'$ are the transition relations in $\mathcal{P}$ and $\mathcal{P}'$ respectively. If there is a simulation pair between $\mathcal{P}$ and $\mathcal{P}'$, we say that $\mathcal{P}'$ simulates $\mathcal{P}$, or $\mathcal{P} \preceq \mathcal{P}'$.

According to (a) the initial state in $\mathcal{P}$ has to be matched by the initial state in $\mathcal{P}'$. Similarly for (b) and individuals. According to (c) and (d) every temporal and epistemic transition in $\mathcal{P}$ has to be matched by a transition in $\mathcal{P}'$. According to (e) and (f) related states must agree on the sub-language $\mathcal{L}'$.

Any $\forall$ACTLK property is preserved from the simulating QIS $\mathcal{P}'$ to the QIS $\mathcal{P}$ being simulated:

Lemma 1. *Assume that $\mathcal{P}'$ simulates $\mathcal{P}$. For any $\forall$ACTLK-formula $\phi \in \mathcal{L}'$, if $\mathcal{P}' \models \phi$ then $\mathcal{P} \models \phi$.*

Lemma 1 follows directly from the following remark:

$$\text{if } (\mathcal{P}'^{\sigma'}, s') \models \phi, s \simeq s' \text{ and } \sigma(x) \approx \sigma'(x) \text{ then } (\mathcal{P}^{\sigma}, s) \models \phi \tag{3}$$

The proof is by induction on the length of ϕ.

Differently from the propositional level, we can define a strengthening of the notion of simulation, so that also existential formulas are also preserved.

Definition 9 (Simulation$^+$)*:* A simulation$^+$ between $\mathcal{P}$ and $\mathcal{P}'$ is a pair of relations $\simeq \subseteq \mathcal{S} \times \mathcal{S}'$ and $\approx \subseteq V \times V'$ such that:

(a) $\simeq$ and $\approx$ are a simulation pair between $\mathcal{P}$ and $\mathcal{P}'$;

(b) if $a' \in V'$ then there exists $a \in V$ such that $a \approx a'$.

If there is a simulation$^+$ pair between $\mathcal{P}$ and $\mathcal{P}'$, we say that $\mathcal{P}'$ simulates$^+$ $\mathcal{P}$, or $\mathcal{P} \preceq^+ \mathcal{P}'$.

We can now prove the following strengthening of Lemma 1.

Lemma 2. *Assume that $\mathcal{P}'$ simulates$^+$ $\mathcal{P}$. For any ACTLK-formula $\phi \in \mathcal{L}'$, if $\mathcal{P}' \models \phi$ then $\mathcal{P} \models \phi$.*

The result follows from (3), where the inductive case for the existential quantifier makes use of clause (b) in Def. 9.

In the following we focus on simulation$^+$ and the ACTLK-fragment.

4.4.2 Existential Abstraction of QIS

In systems with large state spaces, it is infeasible to verify design requirements by considering all reachable states, even if represented symbolically [BCM$^+$92]. In existential abstraction [CGL94, CGJ$^+$03], one reduces a large, possibly infinite reactive system –referred to as the *concrete* system– into a possibly smaller reactive system –the *abstract* system– by partitioning the system states into equivalence classes. Each equivalence class, called an *abstract state*, forms a state in the abstract system.

Here, we extend existential abstraction to quantified interpreted systems by abstracting each agent i and the quantification domain V separately. Formally, the abstract QIS is defined as a quotient construction as follows. Assume a quantified interpreted system $\mathcal{P}$ over the set Ag of agents and the language $\mathcal{L}$. For each $i \in Ag$ assume the equivalence relations $\equiv_i \subseteq L_i \times L_i$ and $\equiv_i \subseteq ACT_i \times ACT_i$. Further, assume an equivalence relation $\equiv \subseteq V \times V$. For $l \in L_i$, write $[l]$ for the equivalence class of l w.r.t. $\equiv_i$, and $[\alpha]$ for the equivalence class of $\alpha \in ACT_i$ w.r.t. $\equiv_i$. Similarly, $[a]$ is the equivalence class of $a \in V$ w.r.t. $\equiv$. Write $[s]$ for $\langle [s_1], \ldots, [s_n] \rangle$ and write $[\vec{\alpha}]$ for $\langle [\alpha_1], \ldots, [\alpha_n] \rangle$. Finally, let $\mathcal{L}' \subseteq \mathcal{L}$ be a sub-language of $\mathcal{L}$ that does not distinguish between equivalent local states and individuals, i.e.,

(*) for all $P^n \in \mathcal{L}'$, if $s \equiv s'$ and $\vec{a} \equiv \vec{a}'$ then $\vec{a} \in I(P^n, s)$ iff $\vec{a}' \in I(P^n, s')$.

Definition 10 (Quotient QIS)*:* The quotient of $\mathcal{P}$ is the QIS $\mathcal{P}'$ on the set Ag of agents and the sub-language $\mathcal{L}' \subseteq \mathcal{L}$ such that:

1. $V' = \{[a] \mid a \in V\}$;
2. $L'_i = \{[l] \mid l \in L_i\}$;
3. $ACT'_i = \{[\alpha] \mid \alpha \in ACT_i\}$;
4. $P'_i = \{\langle [l], [\alpha] \rangle \mid \langle l, \alpha \rangle \in P_i\}$;
5. $\tau' = \{\langle [s], [\vec{\alpha}], [s'] \rangle \mid \langle s, \vec{\alpha}, s' \rangle \in \tau\}$;
6. $s'_0 = [s_0]$;
7. for all $P^n \in \mathcal{L}'$, $[\vec{a}] \in I'(P^n, [s])$ iff $\vec{a} \in I(P^n, s)$;
8. for all $c \in \mathcal{L}'$, $I'(c) = [I(c)]$.

Note that the interpretation I is well defined by condition (*) on the sub-language $\mathcal{L}'$. Observe that Def. 10 does not specify how the equivalence relations are chosen. This issue is addressed in Section 4.4.3 below. The important property of quotient systems, however, is that specifications are preserved from abstract systems to concrete ones. This is because the abstract system simulates$^+$ the original system.

Lemma 3. *If $\mathcal{P}'$ is a quotient of $\mathcal{P}$, then $\mathcal{P}'$ simulates$^+$ $\mathcal{P}$.*

Proof sketch. We show that the relations $\simeq = \{\langle s, [s] \rangle \mid s \in V\}$ and $\approx = \{\langle a, [a] \rangle \mid a \in V\}$ are a simulation$^+$ pair for $\mathcal{P}$ and $\mathcal{P}'$. Simulation requirements (a) and (b) follow from 6 and 1 in Def. 10 respectively. Requirement (c) follows from 4 and 5; while requirement (d) follows by the definition of equivalence classes. Simulation requirements (e) and (f) follow from 7 and 8 respectively. Finally, the simulation$^+$ requirements (b) in Def. 9 trivially hold. □

Since the abstract system simulates$^+$ the concrete system, design requirements expressed in $ACTLK$ are preserved.

Theorem 1 (Preservation)**.** *Let $\mathcal{P}'$ be a quotient of the QIS $\mathcal{P}$. For any ACTLK-formula ϕ in $\mathcal{L}' \subseteq \mathcal{L}$, if $\mathcal{P}' \models \phi$ then $\mathcal{P} \models \phi$.*

Proof. From Lemma 2 and Lemma 3. □

When applying the theorem, the challenge is to choose suitable equivalence relations $\equiv_i$ on local states and actions. We provide a partial answer in the following section.

4.4.3 Constructive Abstraction

In this section we introduce a methodology for defining the equivalence relations defined in the previous section. In what follows we fix a sub-language $\mathcal{L}'$ of the first-order temporal epistemic language $\mathcal{L}_m$, and a QIS $\mathcal{P}$. We first define equivalences on the set V^n of n-tuples of individuals.

Definition 11 (Equivalence on tuples)*:* Let $s \in \mathcal{S}$ and let $\vec{a}$, $\vec{b}$ be n-tuples in V^n. We say that $\vec{a}$ and $\vec{b}$ are equivalent in s, or $\vec{a} \sim_s \vec{b}$, if for all $P^n \in \mathcal{L}'$, $\vec{a} \in I(P^n, s)$ iff $\vec{b} \in I(P^n, s)$.

We can easily check that the relation $\sim_s$ is indeed an equivalence relation for every $s \in \mathcal{S}$.

Definition 12 (Equivalence on individuals)*:* Let $s \in \mathcal{S}$ and let a, b be individuals in V. We say that a and b are equivalent in s, or $a \equiv_s b$, if for all $\vec{a}$, $\vec{a}' \in V^n$, if $\vec{a}'$ is obtained from $\vec{a}$ by uniformly substituting a with b, then $\vec{a} \sim_s \vec{a}'$.

From the fact that $\sim_s$ is an equivalence relation we can derive that also $\equiv_s$ is an equivalence relation for $s \in \mathcal{S}$.

We now define the equivalence relation on states.

Definition 13 (Equivalence on states)*:* Two states s, $s' \in \mathcal{S}$ are equivalent , or $s \equiv s'$, if for all $P^n \in \mathcal{L}'$, for all $\vec{a} \in V^n$, $\vec{a} \in I(P^n, s)$ iff $\vec{a} \in I(P^n, s')$.

We can now introduce the abstract model obtained from the QIS $\mathcal{P}$.

Definition 14 (Abstract model)*:* Given the QIS $\mathcal{P} = \langle V, s_0, \tau, I\rangle$ we define an abstract model $\mathcal{M}' = \langle V', [s_0], \tau', I'\rangle$ such that:

- $\mathcal{S}' = \{[s] \mid s \in \mathcal{S}\}$;
- for each $[s] \in \mathcal{S}'$, $V'([s]) = \{[a]_s \mid a \in V\}$;
- for any $[s], [s'] \in \mathcal{S}$, $[s] \longrightarrow [s']$ if $s \longrightarrow s'$;
- for $P^n \in \mathcal{L}'$, $\vec{[a]_s} \in I'(P^n, [s])$ iff $\vec{a} \in I(P^n, s)$.

By definition of the individuals in each $V'([s])$ we can prove that the interpretation I' is well-defined and independent from the particular choice of representatives for the equivalence classes in $V'([s])$.

Hereafter we introduce the satisfaction relation for the abstract model.

Definition 15*:* The satisfaction relation $\models$ for $\phi \in \mathcal{L}'$, $[s] \in \mathcal{M}'$, and an assignment σ is defined as in Def. 6, but for the following clauses:

$(\mathcal{M}'^{\sigma}, [s]) \models P^k(\vec{t})$ iff $\langle [I'^{\sigma}(t_1)]_s, \ldots, [I'^{\sigma}(t_k)]_s\rangle \in I'(P^k, [s])$
$(\mathcal{M}'^{\sigma}, [s]) \models \forall x\psi$ iff for all $a \in V([s])$, $(\mathcal{M}'^{\sigma^x_a}, [s]) \models \psi$

Now, we can prove the following result on $\mathcal{M}'$.

Lemma 4. *The abstract model $\mathcal{M}'$ simulates$^+$ $\mathcal{P}$*

Proof sketch. The proof consists in showing that the relations $\simeq = \{\langle s, [s]\rangle \mid s \in \mathcal{S}\}$ and $\approx = \{\langle a, [a]_s\rangle \mid a \in V, s \in \mathcal{S}\}$ are a simulation$^+$ pair for $\mathcal{P}$ and $\mathcal{M}'$, and it is similar to Lemma 3. □

The next result is immediate from Lemmas 2 and 4.

Lemma 5. *For every ACTLK-formula ϕ in $\mathcal{L}'$, if $\mathcal{M}' \models \phi$ then $\mathcal{P} \models \phi$.*

Thus, we have obtained an effective way of constructing the quotient system $\mathcal{M}'$ starting from $\mathcal{P}$ and $\mathcal{L}'$. More in detail, given an ACTLK-formula ϕ to be model checked on $\mathcal{P}$, $\mathcal{L}'$ can be thought of as the sub-language of $\mathcal{L}$ consisting of all predicate letters and constants in ϕ.

Finally, we observe that if a QIS $\mathcal{P}$ has infinitely many states and individuals, then also the abstract model $\mathcal{M}'$ will in general be infinite both in states and individuals. However, the construction above does in some cases generate finite approximations.

Lemma 6. *Given a QIS $\mathcal{P}$, its abstract model $\mathcal{M}'$ has the cardinality reported in the following table:*

$\mathcal{P}$		$\mathcal{M}'$	
$\mathcal{S}$	V	$\mathcal{S}'$	V'
infinite	*infinite*	*infinite*	*infinite*
finite	*infinite*	*finite*	*infinite*
infinite	*finite*	*finite*	*finite*
finite	*finite*	*finite*	*finite*

Each line reports the cardinality of sets $\mathcal{S}$ and V in the QIS $\mathcal{P}$, and the cardinality of sets $\mathcal{S}'$ and V' in the abstract model $\mathcal{M}'$.

Proof. *Direct consequence of definition of $\mathcal{M}'$ from $\mathcal{P}$.* □

We have therefore identified a non-trivial case (infinite $\mathcal{S}$ and finite V) in which the technique presented above generates a feasible model checking problem. Additionally, observe that in the case of finite $\mathcal{S}$ and infinite V if all state interpretations are finite (e.g., I maps states into database instances) we also obtain, as result of the abstraction process, a finite $\mathcal{S}'$ in $\mathcal{M}'$.

4.5 Discussion

In this section we have put forward a computationally-grounded semantics for artifact systems and illustrated its use in the context of a temporal epistemic specification language. This is intended to provide a basis for model checking these systems in combination with suitable abstraction techniques. We have provided a first set of results in this context and shown that, at least in certain cases of interest, finite abstractions of artifact-systems can be obtained. A limitation of the results presented is that not in all cases finite abstractions are generated. In verification this is known to be a very severe challenge and is therefore to be expected to appear here as well. Viable options to surmount these difficulties include the development of techniques such as *data independence* [Wol86] in the context of A-QIS, and the investigation on abstraction schemes similar to [DHPV09], e.g., by relaxing the completeness requirement.

5 Foundations of Relational Artifacts Verification

In this section we study the foundations of artifact-centric systems that use relational databases for their data component. Specifically, we consider several artifacts (fixed in advance) forming a *relational artifact system*, each constituted by a *relational database* evolving over time. To characterize such an evolution, we rely on a very rich notion of lifecycle, directly based on stating

dynamic properties in terms of *intra-artifact* and *inter-artifact dynamic constraints*. We express such constraints, and other dynamic properties of interest, in a suitable variant of μ-calculus, one of the most expressive temporal logics used in verification [LPP70, Eme96].

We consider *processes* over artifacts constituted by a set of *actions* (aka atomic tasks, atomic services) and a set of *condition-action rules*, which specify when such actions can be executed. The action specification is possibly the most characterizing part of our framework. Following [CGRR10], actions are specified in terms of preconditions and postconditions on artifacts' databases. Such a specification is strongly influenced by the notion of *mappings* in the recent literature on data exchange and data integration [FKMP05, Len02]. In a nutshell, our action specification considers the current state of the database, and the one obtained by executing an action as two databases related through a set of mappings. In the literature, mappings typically establish correspondences between conjunctive queries, also called tuple-generating dependencies (TGDs) in the database jargon [AHV95]. However here, differently from [CGRR10], we do use negation and more generally full first-order queries in defining the preconditions of actions. Technically speaking, this choice requires us to abandon the theory of conjunctive queries and homomorphisms at the base of the results in [CGRR10, FKMP05, Len02].

We are interested in two main reasoning tasks. The first one is *conformance of a process to an artifact system*, which consists in checking whether a given process generates the correct lifecycle for the various artifacts and, more generally, whether it satisfies all intra-artifact and inter-artifact constraints. The second reasoning task is *process verification*, that is checking whether a process (over an artifact system) verifies general dynamic properties of interest. Both these reasoning tasks in principle can be based on model checking, though, in our setting, one has to deal with potentially infinite states.

We show that both reasoning tasks are undecidable even for very simple artifact systems and processes. We then introduce a very interesting class of processes for which decidability is granted. We call such processes *weakly acyclic*, since they satisfy a condition analogous to weak acyclicity of a set of mappings in data exchange [FKMP05]. Under such a restriction, we are guaranteed that the number of new objects introduced by the execution of actions is finite, and hence, the whole process is finite-state.

The rest of the section is organized as follows. Sections 5.1, 5.2, and 5.3 introduce the framework and illustrate it through a running example. Section 5.4 reports the undecidability of conformance and verification in our framework even for simple cases. Section 5.5 introduces the notion of weakly acyclic processes, and shows that such a restriction makes both conformance and verification decidable. Section 5.6 discusses briefly further works.

5.1 Relational Artifacts Systems

In this section, we start the description of our framework by introducing relational artifact systems. In the following, we assume the reader to be familiar with standard relational databases, and their connection with first-order logic (FOL). In particular, queries are seen as (possibly open) FOL formulas. Also, we consider as special FOL queries conjunctive queries (CQs), i.e., formulas formed only by conjunctions and existential quantifications, and their unions (UCQs) [AHV95].

A *relational artifact systems* RAS is constituted by a set of artifacts, each formed by a relational database evolving over time under restrictions imposed by certain dynamic constraints. We deal with two types of constraints: the *intra-artifact dynamic constraints*, that involve each artifact in isolation, and the *inter-artifact dynamic constraints*, taking into account relations

between artifacts. In this section we introduce such systems.

5.1.1 Relational artifact

A relational artifact is a relational database evolving over time. Hence, it is characterized by the usual notions of *database schema*, giving the structure of the database, and *database instance*, detailing the actual data contained in it, and it is furthermore augmented by a set of *intra-artifact dynamic constrains*. These are temporal constraints expressed in the temporal logic $\mu\mathcal{L}$ introduced later, which allows us to express various constraints over the database: we can assert the usual ones, such as inclusion dependencies, which now become safety temporal constrains, and also what is typically called the *artifact lifecycle*, namely, dynamic constrains on the sequencing of configurations the database may pass through. More formally, a *relational artifact* is a tuple $A = \langle \boldsymbol{R}, I_0, \Phi \rangle$ where:

- $\boldsymbol{R} = \{R_1, \ldots, R_n\}$ is a database schema, constituted by a set of relation schemas;
- I_0 is a database instance, compliant with the schema $\boldsymbol{R}$, that represents the initial state of the artifact;
- Φ is a $\mu\mathcal{L}$ formula over $\boldsymbol{R}$ constituted by the conjunction of all intra-artifact dynamic constraints of A.

Notice that if we project the dynamic formula Φ over the initial artifact instance I_0, we may get (depending on the structure of Φ) static, i.e., local, constraints on I_0. From now on, we assume to deal with *well formed artifacts*, namely, artifacts whose initial instance satisfies such local constraints.

5.1.2 Relational artifact system

A relational artifact system is composed by several relational artifacts in execution at the same time, each consisting of a database and a set of intra-artifact dynamic constraints. The dynamic interaction between them is regulated through additional constraints, also expressed in $\mu\mathcal{L}$, which we call *inter-artifact-dynamic constraints*.

In the following, we make the assumption that artifacts cannot be created or destroyed during the evolution of the system. Under it, we get quite interesting undecidability and decidability results. We are indeed very interested in dropping these limitations in future works, starting from the results presented here. For this reason we start with a finite set of artifacts, and over the whole evolution of the system these will remain the only ones of interest. If an artifact has a terminating lifecycle it becomes dormant, but it will persist in the system.

Formally, an *artifact system* is a pair $\mathcal{A} = \langle \{A_1, \ldots, A_n\}, \Phi_{inter} \rangle$, where $\{A_1, \ldots, A_n\}$ is the finite set of artifacts of the system (each with its own database and intra-artifact dynamic constraints expressed in $\mu\mathcal{L}$), and Φ_{inter} is a $\mu\mathcal{L}$ formula expressing the conjunction of inter-artifact dynamic constraints. To distinguish relations of various artifacts in $\mathcal{A}$, we use the usual *dot notation* of object-orientation, hence, a relation R_j of artifact A_i of $\mathcal{A}$ is denoted by $A_i.R_j$. When clear from the context, we drop the artifact A_i and we use R_j for the relation. We denote by $\mathcal{I}_0$ the disjoint union of all initial instances of the artifacts in $\mathcal{A}$, i.e., $\mathcal{I}_0 = \bigcup_{i=1,\ldots,n} I_{0,i}$. More generally, $\mathcal{I}$ represents the instance obtained by the (disjoint) union of the current instances of each artifact in $\mathcal{A}$.

Given a database instance I, we denote by $\mathcal{C}_I$ the active domain of I, i.e., the set of individuals (typically constants) appearing in I. Hence, the active domain of $\mathcal{I}_0$ is $\mathcal{C}_{\mathcal{I}_0}$, which is made up by all constants appearing in the initial instances of the various artifacts in $\mathcal{A}$.

Notice that, while artifact systems evolve over time, they do not include a predefined mechanism for progression. Progression is due to the execution of actions, tasks, or services over the system, according to a given process that we will introduce later on. Here it is sufficient to assume that a progression mechanism exists, and its execution results in moving from the initial state, given by the instance $\mathcal{I}_0$, to the next one, and so on.

In this way we build a *transition system* [CGP99] $\mathfrak{A}$, whose states represent possible system instances, and each transition an atomic step in the progression mechanism (whatever it is). In principle, we can model-check such a transition system to verify dynamic properties [CGP99], that is exactly what we are going to do next. However, one has to consider that, in general, $\mathfrak{A}$ is infinite, hence the classical results on model checking [CGP99, Eme96], which are developed for finite transition systems, do not apply. The main goal of this work is to find interesting conditions under which such a transition system is finite.

Example 1*:* We model the process of purchasing items within a company. In particular, when a company's employee, that assumes the role of a *requester*, wants to purchase some items, he has to turn to a *buyer*, also internal to the company, who is responsible for purchasing such items from external *suppliers*. In our scenario, we have five actors: two requesters (Bob and Alice), a buyer (Trudy) and two suppliers (SupplierA and SupplierB). The whole purchasing process works as follows: in a first phase, the requester has to fill a so-called *requisition order* with some *line items* chosen from a catalogue. In our simple example the catalogue contains only a monitor, a mouse and a keyboard. Once the requester has completed this process, he sends the order to the buyer, which extracts the line items from it, and purchases each of them separately. In particular, the buyer groups together into a *procurement order* line items (belonging to a requisition order) that will be purchased from a particular supplier. As a result of this phase, we get different procurement orders, each containing line items that the buyer requests from a single supplier. Then the supplier ships back to the buyer the items included in the procurement order he received, and finally, the items are delivered to the original requester. Of course, we can have many orders processed simultaneously in the system, although we will impose some restrictions.

In this example, we consider the relational artifact system $\mathcal{A} = \langle \{\mathsf{ReqOrders}, \mathsf{ProcOrders}\}, \Phi_{inter} \rangle$ containing two relational artifacts, holding all relevant data about requisition orders and procurement orders in the system.

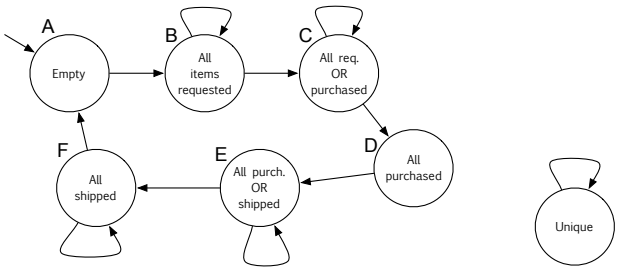
$\mathsf{ReqOrders} = \langle \boldsymbol{R}_{RO}, I_{0,RO}, \Phi_{RO} \rangle$

- $\boldsymbol{R}_{RO} = \{$ $\mathsf{RO}(\mathit{RoCode}, \mathit{ReqName}, \mathit{BuName})$, $\mathsf{ROItem}(\mathit{RoCode}, \mathit{ProdName}, \mathit{Status})$, $\mathsf{Requester}(\mathit{ReqName})$, $\mathsf{LineItem}(\mathit{ProdName}, \mathit{Price})$, $\mathsf{Buyer}(\mathit{BuName})$, $\mathsf{Status}(\mathit{StatusName})$ $\}$

  A requisition order is meant to hold the data associated to every *pending* requisition order: indeed, as soon as the items are delivered to the corresponding requester, each information associated to them is removed from the system. Relation $\mathsf{RO}(\mathit{RoCode}, \mathit{ReqName}, \mathit{BuName})$ holds basic information associated to a single order i.e., order's code and both requester's and buyer's names. The requested items are kept in the relation $\mathsf{ROItem}(\mathit{RoCode}, \mathit{ProdName}, \mathit{Status})$, whose attribute *Status* keeps track of the status of each line item included in the order (it can be either requested, purchased or shipped). Relations $\mathsf{Requester}(\mathit{ReqName})$, $\mathsf{LineItem}(\mathit{ProdName}, \mathit{Price})$, $\mathsf{Buyer}(\mathit{BuName})$ and $\mathsf{Status}(\mathit{StatusName})$ are included in the schema for technical convenience; in particular, the relation Status is needed in order to easily bind values of the attribute *Status* of each line item in an order.

- $I_{0,RO} = \{$ $\mathsf{Requester}(\mathsf{Bob})$, $\mathsf{Requester}(\mathsf{Alice})$, $\mathsf{Buyer}(\mathsf{Trudy})$, $\mathsf{Status}(\mathsf{requested})$, $\mathsf{Status}(\mathsf{purchased})$, $\mathsf{Status}(\mathsf{shipped})$, $\mathsf{LineItem}(\mathsf{keyboard}, 20)$, $\mathsf{LineItem}(\mathsf{mouse}, 10)$, $\mathsf{LineItem}(\mathsf{monitor}, 200)$ $\}$

  According to the previous description of this example scenario, in the initial instance we only have



(a) (b)

Figure 2 – Informal representation of *dynamic* intra-artifact constraints

data referred to existing requesters, buyers, suppliers and the catalogue, featuring three line items. There are no pending orders.

- As for intra-artifact constraints, here we only give an intuition of what will be presented formally later. We want to trace the status of an ordered line item through the attribute ROItem.*Status*, so we express constraints on the evolutions of all orders in the system by relying on this attribute, as informally depicted in Figure 2(a). Intuitively, at the beginning, we do not have any order placed by requesters: in the current situation (henceforth called *phase*) the relations RO and ROItem are empty [A]. As orders are placed, and new requisition orders are created, we will have a phase in which all currently pending orders have status requested [B], and such condition will hold until, *eventually*, some item in such status will be purchased by the buyer by creating procurement orders to send to suppliers, hence changing status to purchased. At this point, we will be in a phase such that all items belonging to existing orders are either requested or purchased [C] and finally, at some point, all orders will be purchased [D]. Having all procurement orders sent to suppliers, some of them will be shipped back, i.e., setting the status of corresponding items to shipped [E], and at the end, all of them will be shipped back to the buyer [F]. Finally, as the items are delivered to the requester, they will be removed from the system and the initial condition will be eventually met again. Notice that these constraints impose restrictions on the evolution of the artifact: for instance, we do not allow new requisition orders (for new line items) to be created unless all the existing ones have been purchased [D]. Notice also that we will need a way to force the system to eventually exit self-loops. Moreover, in addition to such *dynamic* constraints, we also have some *static* ones, such as inclusion dependencies.

ProcOrders $= \langle \boldsymbol{R}_{PO}, I_{0,PO}, \Phi_{PO} \rangle$

- $\boldsymbol{R}_{PO} = \{$ PO($PoCode, RoCode, SupName$), POItem($PoCode, RoCode, ProdName$), Supplier($SupName$), LineItem($ProdName, Price$)$\}$.

  Recall that all line items assigned to the same procurement order must belong to the same requisition order. Hence, similarly to requisition orders, a procurement order's schema includes a relation PO($PoCode, RoCode, SupName$) holding its code, the code of the corresponding requisition order and the name of the chosen supplier. Relation POItem($PoCode, RoCode, ProdName$) holds instead the set of line items in each procurement order. Attribute *RoCode* is replicated in this relation for convenience. Supplier($SupName$) keeps the set of existing suppliers whereas LineItem($ProdName, Price$) is the same as the one in requisition order artifact.

- $I_{0,PO} = \{$ LineItem(keyboard, 20), LineItem(mouse, 10), LineItem(monitor, 200), Supplier(SupplierA), Supplier(SupplierB)$\}$.

- In this example we don't want to constrain the dynamic evolution of the artifact so, informally, the only intra-artifact constraints we will consider are those needed for consistency. ∎

$$
\begin{array}{ll}
(\neg\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \Sigma_{\mathfrak{A}} - (\Phi)^{\mathfrak{A}}_{\mathcal{V}} \\
(\Phi_1 \wedge \Phi_2)^{\mathfrak{A}}_{\mathcal{V}} & = (\Phi_1)^{\mathfrak{A}}_{\mathcal{V}} \cap (\Phi_2)^{\mathfrak{A}}_{\mathcal{V}} \\
(\Phi_1 \vee \Phi_2)^{\mathfrak{A}}_{\mathcal{V}} & = (\Phi_1)^{\mathfrak{A}}_{\mathcal{V}} \cup (\Phi_2)^{\mathfrak{A}}_{\mathcal{V}} \\
(\exists x \in \mathcal{C}_{\mathcal{I}_0}.\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \bigcup\{(\Phi)^{\mathfrak{A}}_{\mathcal{V}[x/c]} \mid c \in \mathcal{C}_{\mathcal{I}_0}\} \\
(\forall x \in \mathcal{C}_{\mathcal{I}_0}.\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \bigcap\{(\Phi)^{\mathfrak{A}}_{\mathcal{V}[x/c]} \mid c \in \mathcal{C}_{\mathcal{I}_0}\}
\end{array}
\qquad
\begin{array}{ll}
(Z)^{\mathfrak{A}}_{\mathcal{V}} & = Z\mathcal{V} \subseteq \Sigma_{\mathfrak{A}} \\
(Q)^{\mathfrak{A}}_{\mathcal{V}} & = \{\mathcal{I} \in \Sigma_{\mathfrak{A}} \mid ans(Q\mathcal{V}, \mathcal{I})\} \\
(\Diamond\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \{\mathcal{I} \in \Sigma_{\mathfrak{A}} \mid \exists \mathcal{I}'.\ \mathcal{I} \Rightarrow_{\mathfrak{A}} \mathcal{I}' \text{ and } \mathcal{I}' \in (\Phi)^{\mathfrak{A}}_{\mathcal{V}}\} \\
(\Box\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \{\mathcal{I} \in \Sigma_{\mathfrak{A}} \mid \forall \mathcal{I}'.\ \mathcal{I} \Rightarrow_{\mathfrak{A}} \mathcal{I}' \text{ implies } \mathcal{I}' \in (\Phi)^{\mathfrak{A}}_{\mathcal{V}}\} \\
(\mu Z.\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \bigcap\{\mathcal{E} \subseteq \Sigma_{\mathfrak{A}} \mid (\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z/\mathcal{E}]} \subseteq \mathcal{E}\} \\
(\nu Z.\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \bigcup\{\mathcal{E} \subseteq \Sigma_{\mathfrak{A}} \mid \mathcal{E} \subseteq (\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z/\mathcal{E}]}\}
\end{array}
$$

Figure 3 – Semantics of $\mu\mathcal{L}$ formulas

5.2 Dynamic Constraints Formalism

We turn to the dynamic constraints formalism, used both to specify intra and inter dynamic constraints of artifact systems (including artifact lifecycles) to specify dynamic properties of processes running over relational artifact systems. Several choices are possible: here we focus on a variant of μ-calculus [Eme96], which is one of the most powerful temporal logics, which subsumes both linear time logics, such as LTL and PSL, and branching time logics such as CTL and CTL* [CGP99]. In particular, we introduce a variant of μ-calculus, called $\mu\mathcal{L}$ that conforms with the basic assumption of our formalism, namely the use of range-restricted FOL queries, i.e., open formulas over a fixed set of constants, to talk about the information contained in the instances.

Formally, $\mu\mathcal{L}$ formulas over $\mathcal{A}$ have the form

$$\Phi ::= Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \exists x \in \mathcal{C}_{\mathcal{I}_0}.\Phi \mid \forall x \in \mathcal{C}_{\mathcal{I}_0}.\Phi \mid \Box\Phi \mid \Diamond\Phi \mid \mu Z.\Phi \mid \nu Z.\Phi \mid Z,$$

where Q is a possibly open FOL formula over the relations in the artifacts of $\mathcal{A}$, and Z is a second order predicate variable. The symbols μ and ν can be considered as quantifiers, and we make use of the notions of scope, bound and free occurrences of variables, closed formulas, etc. Definitions of these notions are as in FOL, treating μ and ν as quantifiers. In fact, we are interested only in closed formulas as specifications of temporal properties to verify.

For formulas of the form $\mu Z.\Phi$ and $\nu Z.\Phi$, we require the *syntactic monotonicity* of Φ wrt Z: every occurrence of the variable Z in Φ must be within the scope of an even number of negation signs. In $\mu\mathcal{L}$, given the requirement of syntactic monotonicity, the least fixpoint $\mu Z.\Phi$ and the greatest fixpoint $\nu Z.\Phi$ always exist.

To define the meaning of a $\mu\mathcal{L}$ formula over an artifact system, we resort to transition systems. Let $\mathfrak{A}$ be a transition system generated by a given progression mechanism over the artifact system $\mathcal{A}$. We denote by $\Sigma_{\mathfrak{A}}$ the states of $\mathfrak{A}$, and by $\mathcal{C}_{\mathfrak{A}}$ all terms (which are in general infinite) occurring in any state of $\mathfrak{A}$. Notice that trivially $\mathcal{C}_{\mathcal{I}_0} \subseteq \mathcal{C}_{\mathfrak{A}}$.

Let $\mathcal{V}$ be a predicate and individual variable valuation on $\mathfrak{A}$, i.e., a mapping from the predicate variables Z to subsets of the states $\Sigma_{\mathfrak{A}}$, and from individual variables to constants in $\mathcal{C}_A$. Then, we assign meaning to $\mu\mathcal{L}$ formulas by associating to $\mathfrak{A}$ and $\mathcal{V}$ an *extension function* $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$, which maps $\mu\mathcal{L}$ formulas to subsets of $\Sigma_{\mathfrak{A}}$. The extension function $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$ is defined inductively as shown in Figure 3, where $Q\mathcal{V}$ (resp., $Z\mathcal{V}$) denotes the application of variable valuation $\mathcal{V}$ to query Q (resp., variables Z), and $ans(Q\mathcal{V}, \mathcal{I})$ denotes the result of evaluating the (boolean) query $Q\mathcal{V}$ over the instance $\mathcal{I}$. Moreover, $\mathcal{I} \Rightarrow_{\mathfrak{A}} \mathcal{I}'$ holds iff the progression mechanism allows to progress from $\mathcal{I}$ to $\mathcal{I}'$.

Intuitively, the extension function $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$ assigns to the various $\mu\mathcal{L}$ constructs the following meanings: The boolean connectives have the expected meaning, while (individual) quantification

involving transitions from some state to the next is restricted to constants of $\mathcal{C}_{I_0}$. The extension of $\Diamond\Phi$ consists of the states $\mathcal{I}$ such that for *some* state $\mathcal{I}'$ with $\mathcal{I} \Rightarrow_{\mathfrak{A}} \mathcal{I}'$, we have that Φ holds in $\mathcal{I}'$, while the extension of $\Box\Phi$ consists of the states $\mathcal{I}$ such that for *all* states $\mathcal{I}'$ with $\mathcal{I} \Rightarrow_{\mathfrak{A}} \mathcal{I}'$, we have that Φ holds in $\mathcal{I}'$. The extension of $\mu Z.\Phi$ is the *smallest subset* $\mathcal{E}_\mu$ of $\Sigma_{\mathfrak{A}}$ such that, assigning to Z the extension $\mathcal{E}_\mu$, the resulting extension of Φ is contained in $\mathcal{E}_\mu$. That is, the extension of $\mu Z.\Phi$ is the *least fixpoint* of the operator $(\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z/\mathcal{E}]}$ (here $\mathcal{V}[Z/\mathcal{E}]$ denotes the predicate valuation obtained from $\mathcal{V}$ by forcing the valuation of Z to be $\mathcal{E}$). Similarly, the extension of $\nu X.\Phi$ is the *greatest subset* $\mathcal{E}_\nu$ of $\Sigma_{\mathfrak{A}}$ such that, assigning to X the extension $\mathcal{E}_\nu$, the resulting extension of Φ contains $\mathcal{E}_\nu$. That is, the extension of $\nu X.\Phi$ is the *greatest fixpoint* of the operator $(\Phi)^{\mathfrak{A}}_{\mathcal{V}[X/\mathcal{E}]}$. When Φ is a closed formula, $(\Phi)^{\mathfrak{A}}_{\mathcal{V}}$ does not depend on $\mathcal{V}$, and we denote it by $\Phi^{\mathfrak{A}}$.

We say that a closed $\mu\mathcal{L}$ formula Φ holds for $\mathfrak{A}$, denoted as $\mathfrak{A} \models \Phi$, iff $\mathcal{I}_0 \in \Phi^{\mathfrak{A}}$. We call *model checking* verifying whether $\mathfrak{A} \models \Phi$ holds.

Example 2 (Continues from Example 1)*:* Now that we have defined our constraints formalism, we are in the position to express the constraints informally discussed in Example 1.

For ReqOrders, we first define formulas corresponding to the phases of the diagram in Figure 2(a):

$$\begin{array}{rcl}
\psi_A & = & \neg\exists x,y,z.\mathsf{ROItem}(x,y,z) \\
\psi_B & = & \forall x,y,z.(\mathsf{ROItem}(x,y,z) \rightarrow z = \mathsf{requested}) \wedge \exists x,y.\mathsf{ROItem}(x,y,\mathsf{requested}) \\
\psi_C & = & \forall x,y,z.(\mathsf{ROItem}(x,y,z) \rightarrow (z = \mathsf{requested} \vee z = \mathsf{purchased})) \wedge \\
 & & \exists x,y.\mathsf{ROItem}(x,y,\mathsf{requested}) \wedge \exists x,y.\mathsf{ROItem}(x,y,\mathsf{purchased}) \\
\psi_D & = & \forall x,y,z.(\mathsf{ROItem}(x,y,z) \rightarrow z = \mathsf{purchased}) \wedge \exists x,y.\mathsf{ROItem}(x,y,\mathsf{purchased}) \\
\psi_E & = & \forall x,y,z.(\mathsf{ROItem}(x,y,z) \rightarrow (z = \mathsf{purchased} \vee z = \mathsf{shipped})) \wedge \\
 & & \exists x,y.\mathsf{ROItem}(x,y,\mathsf{purchased}) \wedge \exists x,y.\mathsf{ROItem}(x,y,\mathsf{shipped}) \\
\psi_F & = & \forall x,y,z.(\mathsf{ROItem}(x,y,z) \rightarrow z = \mathsf{shipped}) \wedge \exists x,y.\mathsf{ROItem}(x,y,\mathsf{shipped}).
\end{array}$$

Then, the dynamic constraints of ReqOrders are captured by the formula

$$\Phi_{RO} = \psi_A \wedge \nu Z.(\textstyle\bigwedge_{i=1,\ldots,11} \Phi_i \wedge \Box Z).$$

It requires that in the initial state of $\mathfrak{A}$ there are not any items included in any pending order, i.e., the relation ROItem is empty, and that all formulas Φ_i listed below hold in every state. Each of Φ_1 to Φ_6 corresponds to a single transition as in Figure 2(a), expressing the constraint that the artifact remains in its current phase *until* it reaches the following one, also requiring that such a phase is eventually reached in a finite number of steps, and that no other phase is reached until then:

$$\begin{array}{rcl@{\qquad}rcl}
\Phi_1 & = & \psi_A \rightarrow \mu Z.(\psi_B \vee (\psi_A \wedge \Box Z)) & \Phi_4 & = & \psi_D \rightarrow \mu Z.(\psi_E \vee (\psi_D \wedge \Box Z)) \\
\Phi_2 & = & \psi_B \rightarrow \mu Z.(\psi_C \vee (\psi_B \wedge \Box Z)) & \Phi_5 & = & \psi_E \rightarrow \mu Z.(\psi_F \vee (\psi_E \wedge \Box Z)) \\
\Phi_3 & = & \psi_C \rightarrow \mu Z.(\psi_D \vee (\psi_C \wedge \Box Z)) & \Phi_6 & = & \psi_F \rightarrow \mu Z.(\psi_A \vee (\psi_F \wedge \Box Z)).
\end{array}$$

The remaining formulas express static constraints, specifically inclusion dependencies and range restrictions:

$$\begin{array}{rcl}
\Phi_7 & = & \forall x,y,z.(\mathsf{ROItem}(x,y,z) \rightarrow \exists u,v.\mathsf{RO}(x,u,v)) \\
\Phi_8 & = & \forall x,y,z.(\mathsf{ROItem}(x,y,z) \rightarrow \mathsf{Status}(z)) \\
\Phi_9 & = & \forall x.(\mathsf{Status}(x) \rightarrow (x = \mathsf{requested} \vee x = \mathsf{purchased} \vee x = \mathsf{shipped})) \\
\Phi_{10} & = & \forall x,y,z.(\mathsf{ROItem}(x,y,z) \rightarrow \exists w\mathsf{LineItem}(y,w)) \\
\Phi_{11} & = & \forall x,y,z.(\mathsf{RO}(x,y,z) \rightarrow (\mathsf{Requester}(y) \wedge \mathsf{Buyer}(z)).
\end{array}$$

For ProcOrders, we just need to express some static specifications over instances. Hence, Φ_{PO} is the conjunction of the following formulas, expressing inclusion dependency constraints:

$$\begin{array}{l}
\nu Z.(\forall x,y,z.(\mathsf{PO}(x,y,z) \rightarrow \mathsf{Supplier}(z)) \wedge \Box Z) \\
\nu Z.(\forall x,y,z.(\mathsf{POItem}(x,y,z) \rightarrow \exists u.\mathsf{PO}(x,y,u)) \wedge \Box Z) \\
\nu Z.(\forall x,y,z.(\mathsf{POItem}(x,y,z) \rightarrow \exists u.\mathsf{LineItem}(u,z)) \wedge \Box Z).
\end{array}$$

Finally, the set of inter-artifact dynamic constraints Φ_{inter} is the conjunction of the following formulas:

$$\nu Z.(\forall x, y.(\mathsf{ReqOrders.LineItem}(x, y) \leftrightarrow \mathsf{ProcOrders.LineItem}(x, y)) \wedge \Box Z)$$
$$\nu Z.(\forall x, y, z.(\mathsf{POItem}(x, y, z) \rightarrow \mathsf{ROItem}(y, z, \mathsf{purchased})) \wedge \Box Z)$$
$$\nu Z.(\forall x, y, z.(\mathsf{PO}(x, y, z) \rightarrow \exists w, k.\mathsf{RO}(y, w, k)) \wedge \Box Z).$$

The first formula requires that the LineItem relations in both artifacts have the same set of tuples, the second one that every item belonging to a procurement order is also included in some requisition order, and the third one that every procurement order corresponds to a requisition order. ∎

5.3 Processes over Artifact Systems

We now concentrate on progression mechanisms for relational artifact systems. In particular, we specify such a mechanism in terms of one or more *processes* that use actions as atomic steps. *Actions* represent atomic tasks or services that act over the artifacts and make them evolve.

5.3.1 Actions

We give a formal specification of actions in terms of preconditions and postconditions [HR99], inspired by the notion of mapping in the literature on data exchange [Kol05]. However, we generalize such a notion in order to include negation, arbitrary quantification in preconditions, and the generation of new terms, through the use of *Skolem functions* in postconditions. Notice that, while it is conceivable that most of the actions will act on one artifact only, we do not make such a restriction. Indeed our actions are generally inter-artifact, which lets us easily account for synchronisation between artifacts.

An *action* ρ for $\mathcal{A}$ has the form

$$\rho(p_1, \ldots, p_m) : \{e_1, \ldots, e_m\} \qquad \text{where:}$$

- $\rho(p_1, \ldots, p_m)$ is the *signature* of the action, constituted by a name ρ and a sequence $p_1, \ldots, p_m$ of *input parameters* that need to be substituted by constants for the execution of the action, and
- $\{e_1, \ldots, e_m\}$ is a set of effects, called the *effects' specification*.

We denote by σ a (ground) substitution for the input parameters with terms not involving variables. Given such a substitution σ, we denote by $\rho\sigma$ the action with actual parameters. All effects in the effects' specification are assumed to take place simultaneously. Specifically, an *effect* e_i has the form

$$q_i^+ \wedge Q_i^- \rightsquigarrow I_i' \qquad \text{where:}$$

- $q_i^+ \wedge Q_i^-$ is a query whose terms are variables $\vec{x}$, action parameters, and constants from $\mathcal{C}_{\mathcal{I}_0}$. The query q_i^+ is a UCQ, and the query Q_i^-, is an arbitrary FOL formula whose free variables are included in those of q_i^+. Intuitively, q_i^+ selects the tuples to instantiate the effect, and Q_i^- filters away some of them.
- I_i' is a set of facts for the artifacts in $\mathcal{A}$, which includes as terms: terms in $\mathcal{C}_{\mathcal{I}_0}$, input parameters, free variables of q_i^+, and in addition terms formed by applying an arbitrary Skolem function to one of the previous kinds of terms. Such Skolem terms are used as witnesses of values chosen by the external user/environment when executing the action. Notice that different effects can share a same Skolem function.

Given an instance $\mathcal{I}$ of $\mathcal{A}$, an effect e_i as above, and a substitution σ for the parameters of e_i, the effect e_i extracts from $\mathcal{I}$ the set $ans((q_i^+ \wedge Q_i^-)\sigma, \mathcal{I})$ of tuples of terms, and for each such tuple θ asserts the set $I_i'\sigma\theta$ of facts obtained from $I_i'\sigma$ by applying the substitution θ for the free variables of q_i^+. In particular, in the resulting set of facts we may have terms of the form $f(\vec{t})\sigma\theta$ where $\vec{t}$ is a set of terms that may be either free variables in $\vec{x}$, parameters, or terms in $\mathcal{C}_{\mathcal{I}_0}$. We denote by $e_i\sigma(\mathcal{I})$ the overall set of facts, i.e., $e_i\sigma(\mathcal{I}) = \bigcup_{\theta \in ans((q_i^+ \wedge Q_i^-)\sigma, \mathcal{I})} I_i'\sigma\theta$. The overall *effect* of the action ρ with parameter substitution σ over $\mathcal{I}$ is a new instance $\mathcal{I}' = do(\rho\sigma, \mathcal{I}) = \bigcup_{1 \leq i \leq m} e_i\sigma(\mathcal{I})$ for A.

Some observations are in order: *(i)* In the formalization above actions are *deterministic*, in the sense that, given an instance $\mathcal{I}$ of $\mathcal{A}$ and a substitution σ for the parameters of an action ρ, there is a *single* instance $\mathcal{I}'$ that is obtained as the result of executing ρ in $\mathcal{I}$. *(ii)* The effects of an action are naturally a form of update of the previous state, and not of belief revision [KM91]. That is, we never learn new facts on the state in which an action is executed, but only on the state resulting from the action execution. *(iii)* We do not make any persistence (or frame) assumption in our formalization [Rei01]. In principle at every move we substitute the whole old state, i.e., instance, $\mathcal{I}$, with a new one, $\mathcal{I}'$. On the other hand, it should be clear that we can easily write effect specifications that *copy* big chunks of the old state into the new one. For example, $R_i(\vec{x}) \rightsquigarrow R_i(\vec{x})$ copies the entire set of assertions involving the relation R_i.

5.3.2 Processes

Essentially processes are (possibly nondeterministic) programs that use artifacts in $\mathcal{A}$ to store their (intermediate and final) computation results, and use actions of the artifacts in $\mathcal{A}$ as atomic instructions. We assume that at every time the current instance $\mathcal{I}$ can be arbitrarily queried through the query answering services, while it can be updated only through the actions of the artifacts in $\mathcal{A}$. Notice that, while we require the execution of actions to be sequential, we do not impose any such constraints on processes, which in principle can be formed by several concurrent branches, including fork, join, and so on. Concurrency is to be interpreted by interleaving, as often done in formal verification [CGP99, Eme96]. There can be many ways to provide the control flow specification for processes for $\mathcal{A}$. Here we adopt a very simple rule-based mechanism. Notice, however, that our results can be immediately generalized to any process formalism whose processes control flow is finite-state. Notice also that the transition system associated to a process over an artifact might not be finite-state, since its state is formed by both the *control flow* state of the process and the *data* in the artifact system, which are in general unbounded.

Formally, a process Π over a relational artifact system $\mathcal{A}$ is a pair $\langle \boldsymbol{\rho}, \boldsymbol{\pi} \rangle$, where $\boldsymbol{\rho}$ is a finite set of actions and $\boldsymbol{\pi}$ is a finite set of condition-action rules.

A *condition-action rule* π in $\boldsymbol{\pi}$ is an expression of the form

$$Q \mapsto \rho,$$

where ρ is an action in $\boldsymbol{\rho}$ and Q is a FOL formula over artifacts' relations whose free variables are exactly the parameters of ρ, and whose other terms can be either quantified variables or terms in $\mathcal{C}_{\mathcal{I}_0}$. Such a rule has the following semantics: for each tuple σ for which condition Q holds, the action ρ with actual parameters σ *can* be executed. If ρ has no parameters then Q will be a boolean formula. Observe that processes don't force the execution of actions but constrain them: the user of the process will be able to choose any of the actions that the rules forming the process allow.

The *execution* of a process Π over a relational artifact system $\mathcal{A}$ is defined as follows: we start from $\mathcal{I}_0$, and for each rule $Q \mapsto \rho$ in Π, we evaluate Q, and for each tuple σ returned, we execute $\rho\sigma$, obtaining a new instance $\mathcal{I}' = do(\rho\sigma, \mathcal{I}_0)$, and so on. In this way we build a *transition system* $\Upsilon(\Pi, \mathcal{A})$ whose states represent possible system instances, and where each transition represents the execution of an instantiated action that is allowed according to the process. A transition $\mathcal{I} \Rightarrow_{\Upsilon(\Pi,\mathcal{A})} \mathcal{I}'$ holds iff there exists a rule $Q \mapsto \rho$ in Π such that there exists a $\sigma \in ans(Q, \mathcal{I})$ and $\mathcal{I}' = do(\rho\sigma, \mathcal{I})$. That is, there exist a rule in Π that can fire on $\mathcal{I}$ and produce an instantiated action $\rho\sigma$, which applied on $\mathcal{I}$, results in $\mathcal{I}'$.

The transition system $\Upsilon(\Pi, \mathcal{A})$ captures the behavior of the process Π over the whole system $\mathcal{A}$. We are interested in formally verifying properties of processes over artifact-based systems, in particular we are interested in *conformance* and *verification*, defined as follows:

Conformance. Given a process Π and an artifact system $\mathcal{A}$, the process is said to be acceptable if it fulfills all intra-artifact and inter-artifact dynamic constraints. In this case, we say that Π *conforms to* $\mathcal{A}$. In order to formally check *conformance*, we can resort to model checking and verify that:

$$\Upsilon(\Pi, \mathcal{A}) \models \Phi_{inter} \wedge \bigwedge_{i=1,\ldots,n} \Phi_i.$$

Verification. Apart from intra-artifacts and inter-artifact dynamic constraints, we are interested in other dynamic properties of the process over the artifact system. We say that a process Π over an artifact system $\mathcal{A}$ *verifies* a dynamic property Φ expressed in $\mu\mathcal{L}$ if

$$\Upsilon(\Pi, \mathcal{A}) \models \Phi.$$

It becomes evident that model checking of the transition system $\Upsilon(\Pi, \mathcal{A})$ generated by a process over an artifact system is the critical form of reasoning needed in our framework. We are going to study such a reasoning task next.

Example 3 (Continues from Example 2)*:* We consider a process $\Pi = \langle \boldsymbol{\rho}, \boldsymbol{\pi} \rangle$ constituted by the following actions $\boldsymbol{\rho}$ and conditions-action rules $\boldsymbol{\pi}$. When specifying an action, we will use $[\ldots]$ to delimit each of the two parts q_i^+ and Q_i^- of the formula $q_i^+ \wedge Q_i^-$ in the left-hand side of an effect specification. Note that in such a formula the part corresponding to Q_i^- might be missing.

Actions. The set $\vec{\rho}$ of actions is the following. Action $\mathsf{RequestItem}(r, i, b)$ is used by the requester r to request a new line item i to buyer b. Such an action results in adding i to the requisition order of r. Notice that the *RoCode* denoting the requisition order is computed as a function $f(r, b)$: performing this action multiple times for the same requester and buyer will result into adding line items to the same requisition order.

$$\begin{aligned}\mathsf{RequestItem}(r,i,b): \ \{\ & [\exists w.(\mathsf{Requester}(r) \wedge \mathsf{LineItem}(i,w) \wedge \mathsf{Buyer}(b))] \rightsquigarrow \\ & \{\mathsf{RO}(f(r,b), r, b),\ \mathsf{ROItem}(f(r,b), i, \mathsf{requested})\}, \\ & \mathit{CopyAll}\ \}\end{aligned}$$

Action $\mathsf{Purchase}(r, i, b, s)$ is used by buyer b for purchasing an item i belonging to requisition order r from supplier s, thus creating (or updating) procurement orders (i.e., the relation ProcOrders.PO) and updating the status of the corresponding items kept by the relation ReqOrders.ROItem. Again, notice that *PoCode*, whose value is denoted below as $g(r, b, s)$, is not a function of the item i passed as parameter. By writing $\mathit{CopyAll} \setminus \mathsf{ROItem}$ we denote the copy of all relations except ROItem.

$$\begin{aligned}\mathsf{Purchase}(r,i,b,s): \ \{\ & [\exists w.\mathsf{RO}(r,w,b) \wedge \mathsf{ROItem}(r,i,\mathsf{requested}) \wedge \mathsf{Supplier}(s)] \rightsquigarrow \\ & \{\mathsf{PO}(g(r,b,s), r, s),\ \mathsf{POItem}(g(r,b,s), r, i), \\ & \ \mathsf{ROItem}(r, i, \mathsf{purchased})\}, \\ & [\mathsf{ROItem}(x,y,z)] \wedge [\neg \mathsf{ROItem}(r,i,\mathsf{requested})] \rightsquigarrow \\ & \{\mathsf{ROItem}(x,y,z)\}, \\ & \mathit{CopyAll} \setminus \mathsf{ROItem}\ \}\end{aligned}$$

The following actions are used to ship all items included in a given procurement order p, and to deliver items belonging to a requisition order r to the corresponding requester, respectively. Notice that the first avoids copying all facts concerning p whereas the latter does the same with all facts related to r.

$$\begin{aligned}
\mathsf{Ship}(p): \quad \{\ & [\mathsf{POItem}(p,x,y) \wedge \exists z.\mathsf{ROItem}(x,y,z)] \rightsquigarrow \{\mathsf{ROItem}(x,y,\mathsf{shipped})\},\\
& [\mathsf{POItem}(x,y,z)] \wedge [\neg\mathsf{POItem}(p,y,z)] \rightsquigarrow \{\mathsf{POItem}(x,y,z)\},\\
& [\mathsf{PO}(x,y,z)] \wedge [\neg\mathsf{PO}(p,y,z)] \rightsquigarrow \{\mathsf{PO}(x,y,z)\},\\
& \mathit{CopyAll} \setminus (\mathsf{POItem} \text{ and } \mathsf{PO})\ \}\\
\mathsf{Deliver}(r): \quad \{\ & [\mathsf{ROItem}(x,y,z)] \wedge [\neg\mathsf{ROItem}(r,y,z)] \rightsquigarrow \{\mathsf{ROItem}(x,y,z)\},\\
& [\mathsf{RO}(x,y,z)] \wedge [\neg\mathsf{RO}(r,y,z)] \rightsquigarrow \{\mathsf{RO}(x,y,z)\},\\
& \mathit{CopyAll} \setminus (\mathsf{ROItem} \text{ and } \mathsf{RO})\ \}
\end{aligned}$$

Condition-action rules. In each condition-action rule of our process, we instantiate the parameters passed to the action, while simply checking that they are meaningful, i.e., that they are in the current instance. Hence:

$$\begin{aligned}
\boldsymbol{\pi} = \{\ & \exists x.(\mathsf{Requester}(r) \wedge \mathsf{LineItem}(i,x) \wedge \mathsf{Buyer}(b)) \mapsto \mathsf{RequestItem}(r,i,b),\\
& \exists x.(\mathsf{RO}(r,x,b) \wedge \mathsf{ROItem}(r,i,\mathsf{requested}) \wedge \mathsf{Supplier}(s)) \mapsto \mathsf{Purchase}(r,i,b,s),\\
& \exists x,y.\mathsf{PO}(p,x,y) \mapsto \mathsf{Ship}(p),\\
& \exists x,y.\mathsf{RO}(r,x,y) \mapsto \mathsf{Deliver}(r)\ \}
\end{aligned}$$

We close our example by observing that the process we have specified conforms to the lifecycle in Example 2. ∎

5.4 Undecidability of Conformance and Verification

Next, we consider conformance and verification over relational artifact systems. We show that they are both undecidable in general, even in simple cases. The undecidability result does not come as a surprise, since the transition system of a process over an artifact system can easily be infinite-state. However, we show that the undecidability holds even in a very simple case.

We consider a relational artifact system of the form $\mathcal{A}_u = \langle\{A\}, \mathsf{true}\rangle$ with $A = \langle\vec{R}, I_0, \mathsf{true}\rangle$. That is $\mathcal{A}_u$ is formed by a single artifact A with no intra-artifact or inter-artifact dynamic constraints. In addition, we consider processes with only one action ρ_u and only one condition-action rule $\mathsf{true} \mapsto \rho_u$ that has a true condition and hence allows the execution of the action ρ_u at every moment. The action ρ_u is without parameters, its effects have the form

$$q_i^+ \rightsquigarrow I_i',$$

where q_i^+ is a CQ (hence without any form of negation and of universal quantification), and it includes *CopyAll* effects. We call these kinds of relational artifact systems and processes *simple*. Next lemma shows that it is undecidable to verify in such cases the $\mu\mathcal{L}$ formula $\mu Z.(q \vee \Diamond Z)$, expressing that there exists a sequence of action executions that leads to an instance where a boolean CQ q holds.

Lemma 7. *Verifying whether the $\mu\mathcal{L}$ formula $\mu Z.(q \vee \Diamond Z)$ holds for a simple process over a simple artifact is undecidable.*

Sketch. We observe that we can use the set of effects of ρ_u to encode a set of tuple-generating dependencies (TGDs) [AHV95]. Hence we can reduce to the above verification problem the

problem of answering boolean CQs in a relational database under a set of TGDs, which is undecidable [BV81]. (In fact, special care is needed because of the use of Skolem terms instead of labeled nulls.) □

Theorem 2. *Conformance checking and verification are both undecidable for processes over relational artifacts systems.*

sketch. Lemma 7 gives us undecidability of verification, already for simple relational artifact systems and processes. To get undecidability of conformance it is sufficient to consider the simple process Π_u over relational artifact systems of the form $\mathcal{A}_{cu} = \langle\{A_c\}, \mathsf{true}\rangle$, with $A_c = \langle\vec{R}, I_0, \mu Z.(q \vee \Diamond Z)\rangle$. Note that $\mathcal{A}_{cu}$ is a variant of simple artifact systems $\mathcal{A}_u$ in which the artifact has as intra-artifact dynamic constraint exactly $\mu Z.(q \vee \Diamond Z)$. The claim follows again from Lemma 7, considering that, by definition, checking conformance of the simple process Π_u wrt $\mathcal{A}_{cu}$ is equivalent to checking whether $\Upsilon(\Pi_u, \mathcal{A}_u) \models \mu Z.(q \vee \Diamond Z)$. □

5.5 Decidability of Weakly Acyclic Processes

Next we tackle decidability, and, inspired by the recent literature on data exchange [Kol05], we isolate a notable case of processes over relational artifact systems for which both conformance and verification are decidable. Our results rely on the possibility of building a special process that we call "positive approximate". For such a process there exists a tight correspondence between the application of an action and a step in the chase of a set of TGDs [AHV95, Kol05]

Given a process $\Pi = \langle\boldsymbol{\rho}, \boldsymbol{\pi}\rangle$, the *positive approximate* of Π is the process $\Pi^+ = \langle\boldsymbol{\rho}^+, \boldsymbol{\pi}^+\rangle$ obtained from Π as follows. For each action ρ in $\boldsymbol{\rho}$, there is an action ρ^+ in $\boldsymbol{\rho}^+$, obtained from ρ by

- removing all input parameters from the signature, and
- substituting each effect $q_i^+ \wedge Q_i^- \rightsquigarrow I_i'$ with the one that uses only the *positive* part of the head of the effect specification, i.e., with $q_i^+ \rightsquigarrow I_i'$.

Note that the variables in q_i^+ that used to be parameters in ρ, become free variables in ρ^+. Then, for each condition-action rule $Q \mapsto \rho$ in $\boldsymbol{\pi}$, there is a rule $\mathsf{true} \mapsto \rho^+$ in $\boldsymbol{\pi}^+$. Hence, Π^+ allows for executing every action at every step.

Now, relying again on the parallelism between chase in data exchange and action execution in artifact systems, we take advantage of the notion of weak acyclicity in data exchange [Kol05] to devise an interesting class of processes which are guaranteed to generate a finite-state transition system, when run over a relational artifact system. This in turn guarantees decidability of conformance and verification.

Let Π be a process over an artifact system $\mathcal{A}$, and $\Pi^+ = \langle\boldsymbol{\rho}^+, \boldsymbol{\pi}^+\rangle$ its positive approximate. We call *dependency graph* of Π^+ the following (edge labeled) directed graph:

Nodes: for every artifact $A = \langle\boldsymbol{R}, I_0, \Phi\rangle$ of $\mathcal{A}$, every relation symbol $R_i \in \boldsymbol{R}$, and every attribute *att* or R_i, there is a node (R_i, att) representing a position;

Edges: for every action ρ^+ of $\boldsymbol{\rho}^+$, every effect $q_i^+(\boldsymbol{t}) \rightsquigarrow I_i'(\boldsymbol{t}', f_1(\boldsymbol{t}_1), \ldots, f_n(\boldsymbol{t}_n))$ of ρ^+ (where for convenience we have made explicit the terms occurring in q_i^+ and I_i', and where consequently $\boldsymbol{t}', \boldsymbol{t}_1, \ldots, \boldsymbol{t}_n \subseteq \boldsymbol{t}$ are either constants or variables), every variable $x \in \boldsymbol{t}$, and every occurrence of x in q_i^+ in position p, there are the following edges:

- for every occurrence of x in I'_i in position p', there is an edge $p \rightarrow p'$;
- for every Skolem term $f_k(t_k)$ such that $x \in \boldsymbol{t}_k$ occurs in I'_i in position p'', there is a *special edge* (i.e., one labeled by $*$) $p \xrightarrow{*} p''$.

We say that Π is *weakly acyclic* if the dependency graph of Π^+ has no cycle going through a special edge.

Intuitively, ordinary edges keeps track of the fact that a value may propagate from position p to position p' in a possible trace. Moreover, special edges keeps track of the fact that a value in position p can be taken as parameter of a Skolem function, thus contributing to the creation of a (not necessarily new) value in any position p''. If a cycle goes through a special edge, then a new value appearing in a certain position may determine the creation of another one, in the same position, later during the execution of actions. Since this may happen again and again, no bound can be put on the number of newly generated Skolem terms, and thus on the number of new values appearing in the instance. Note that the definition allows for cycles as long as they do not include special edges.

Lemma 8. *Let* Π *be a* weakly acyclic *process over a relational artifact system* $\mathcal{A}$ *with initial instance* $\mathcal{I}_0$*, and let* Π^+ *be the positive approximate of* Π*. Then there exists a polinomial in the size of* $\mathcal{I}_0$ *that bounds the size of every instance generated by* Π^+*.*

sketch. The proof follows the line of that in [Kol05] on chase termination for weakly acyclic TGDs. The difference here is that we use Skolem terms and don't have the inflationary behavior of TGDs in applying action effects. However, the key notion of rank used in [Kol05] can still be used to bound the number of terms generated through the Skolem functions. □

Notice that as a direct result of this lemma, the transition system generated by the positive approximate over $\mathcal{A}$ has a number of states that is finite, and in fact at most exponential in the size of the initial instance $\mathcal{I}_0$ of $\mathcal{A}$. Now we show that a similar result holds for the original process Π. The key to this is the following observation that easily follows from the definition of ρ^+ for an action ρ.

Lemma 9. *For every action* ρ *over* $\mathcal{A}$*, instances* $\mathcal{I}_1$*,* $\mathcal{I}_2$ *of* $\mathcal{A}$*, and ground substitution* σ *for the parameters of* ρ*, if* $\mathcal{I}_1 \subseteq \mathcal{I}_2$ *then* $do(\rho\sigma, \mathcal{I}_1) \subseteq do(\rho^+, \mathcal{I}_2)$*.*

We can extend the result above to any sequence of actions, by induction on the length of the sequence. Hence, we get that the instance obtained from the initial instance by executing a sequence of actions of the original process Π is contained in the instance obtained by executing the same sequence of actions of Π^+. From this observation, considering the bound in Lemma 8, we get the desired result for the original process.

Lemma 10. *Let* Π *be a* weakly acyclic *process over a relational artifact system* $\mathcal{A}$ *with initial instance* $\mathcal{I}_0$*. Then there exists a polinomial in the size of* $\mathcal{I}_0$ *that bounds the size of every instance generated by* Π*.*

From this, we obtain our main result.

Theorem 3. *Conformance and verification of* $\mu\mathcal{L}$ *formulas are decidable for weakly acyclic processes over relational artifact systems.*

sketch. From Lemma 10, it follows that the transition system generated by a weakly acyclic process over a relational artifact system $\mathcal{A}$ has a number of states that is at most exponential in the size of the initial instance $\mathcal{I}_0$ of $\mathcal{A}$. The claim then follows from known results on verification of μ-calculus formulas over finite transition systems (see e.g., [Eme96]). □

From the exponential bound on the number of states of the generated transition system mentioned in the proof above, we get not only decidability of verification and conformance, but also an ExpTime upper bound for its computational complexity (assuming a bound on the nesting of fixpoints).

5.6 Discussion

In this section we have looked at foundations of artifact-centric systems, and we have shown that weakly acyclic processes over relational artifacts are very interesting both from a formal point of view, since reasoning on them is decidable, and from a practical point of view, since weak-acyclicity appears to be a quite acceptable restriction, as it essentially amounts to avoiding the infinite accumulation on new relevant data.

Further research can take several directions. First, one can easily focus on different temporal logics for specifying dynamic constraints, such as LTL or CTL. Observe that the results presented here would apply, being μ-calculus more expressive than both LTL and CTL, but certainly they can be refined. Second, we may introduce special equality generating constraints to allow to equate different terms, e.g., a Skolem term and a constant. We are particularly interested in how to extend our decidability result to this case. Also we have assumed that no artifacts are added or destroyed during the execution of a process. We are very interested in overcoming this assumption. Notice that to do so we would need to introduce Skolem terms to denote artifacts, and then extend the notion of weakly acyclic process to block the infinite accumulation of new artifacts. Finally, we are interested in moving from a relational setting to a semantic one, based on ontologies for data access [CDGL$^+$11], believing that similar results apply.

6 Conclusions and Future Work

In this document we have described the contributions of WP2, after one year of work in the ACSI project. We have identified the project requirements for what concerns the main topics of WP2, i.e., verification and synthesis, and we have presented the main achievements obtained, namely:

- a formal model and a specification language for reasoning about temporal-epistemic properties of artifact systems, over which we developed an abstraction methodology that guarantees decidability in some cases of practical interests;
- an actual technique for detecting when the executions of an artifact system are guaranteed to manipulate with only a finite set of new elements, which opens the possibility of adapting standard Model Checking techniques for general (i.e., linear- and branching-time) temporal properties, to artifact systems.

Such contributions represent significant advances in our understanding of the verification and synthesis problems in the artifact context. Notice that although they apparently use different

formal models, the difference is not substantial. This is essentially due to the fact that the contributions propose two different approaches to similar problems, namely: Section 4 defines a general abstraction methodology on a purely formal model for artifact systems, that enables verification of temporal-epistemic properties over data –notice that this requires the presence of *agents* as *subjects* of the knowledge, while Section 5 proposes an actual technique to check that a system satisfies sufficient conditions that guarantee decidability of the verification task for temporal properties.

As already discussed, at this stage, the achievements in verification constitute a basis also for synthesis as, similarly to the propositional case, most of the framework developed for verification can be used (possibly extended) for synthesis purposes, as well.

The main difficulty we encountered, which is also the characterising feature of artifact systems from the verification viewpoint, is the presence of data, its infiniteness, and the relevance of properties over them, which rules out existing approaches, e.g., propositional model checking, and call for suitable first-order generalisation. Therefore in our results, we naturally extend methodologies and techniques developed in the broad area of formal verification and synthesis, including abstraction and verification of database-driven systems [Via09].

With respect to future developments, the work carried out so far leaves many open directions. The work presented in Section 4 constitutes a theoretical framework for reasoning about temporal-epistemic properties of artifact systems. Based on this, actual verification techniques need to be developed. In particular, we expect to be able to verify properties similar to those addressed in Section 5, extended with epistemic modalities, provided the artifact system does not exceed a size bound fixed a priori. Observe this is a different condition than the one checked by the technique presented in Section 5. Indeed, that technique checks whether the number of different values manipulated during each execution is finite, whereas we are interested in bounding the number of different values present at each state, although along an execution the number of manipulated values can be infinite.

As to the contribution described in Section 5, an important issue is the development of *effective* verification techniques. As a matter of fact, although decidable, the verification task turns out extremely complex, namely time-exponential, whereby the need for developing techniques, possibly based on abstraction (used to narrow the search space), that perform well in the practice. Another relevant point concerns richness of the specification language. In particular, verifiable properties do not include the equality symbol, and the use of quantifiers is restricted to local states or to the initial state (possibly combined), while interactions among generic states is not in general possible. Weakening these restrictions would allow for capturing a larger class of properties over artifact systems, and is certainly of interest to the project. Finally, the work currently assumes that no artifact is created or deleted during execution. While this is not as restrictive as it may appear (in fact, an artifact instance corresponds to a whole database), it certainly constitutes a limitation on the class of systems that can be verified. Weakening this restriction requires a major extension on the introduced framework, that will be considered in the next stage.

Bibliography

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49(5):672–713, 2002.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

[AMPS98] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller Synthesis for Timed Automata. In *Proc. IFAC Symposium on System Structure and Control*, pages 469–474, 1998.

[AT04] Rajeev Alur and Salvatore La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Log.*, 5(1):1–25, 2004.

[AVFY00] Serge Abiteboul, Victor Vianu, Bradley S. Fordham, and Yelena Yesha. Relational Transducers for Electronic Commerce. *J. Comput. Syst. Sci.*, 61(2):236–269, 2000.

[BAPM83] Mordechai Ben-Ari, Amir Pnueli, and Zohar Manna. The Temporal Logic of Branching Time. *Acta Informatica*, 20:207–226, 1983.

[BCG+05] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. Automatic Composition of Transition-based Semantic Web Services with Messaging. In *Proc. of VLDB*, pages 613–624, 2005.

[BCJ+97] Anca Browne, Edmund M. Clarke, Somesh Jha, David E. Long, and Wilfredo R. Marrero. An Improved Algorithm for the Evaluation of Fixpoint Expressions. *Theor. Comput. Sci.*, 178(1-2):237–255, 1997.

[BCM+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^20 States and Beyond. *Information and Computation*, 98(2):142–170, 1992.

[BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *Proc. of CONCUR*, pages 135–150, 1997.

[BG04] Achim Blumensath and Erich Grädel. Finite Presentations of Infinite Structures: Automata and Interpretations. *Theory Comput. Syst.*, 37(6):641–674, 2004.

[BHV04] Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract Regular Model Checking. In *Proc. of CAV*, pages 372–386, 2004.

[BJNT00] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular Model Checking. In *Proc. of CAV*, pages 403–418, 2000.

[BL69] J.R. Büchi and L.H. Landweber. Solving sequential conditions by ?nite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.

[BL09] Francesco Belardinelli and Alessio Lomuscio. Quantified epistemic logics for reasoning about knowledge in multi-agent systems. *Artif. Intell.*, 173(9-10):982–1013, 2009.

[Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[Büc62] J.R. Büchi. On a Decision Method in Restricted Second Order Arithmetic. In *Proc. of Int. Congress for Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.

[BV81] Catriel Beeri and Moshe Y. Vardi. The Implication Problem for Data Dependencies. In *Proc. of ICALP'81*, volume 115 of *LNCS*, pages 73–85. Springer, 1981.

[Cau02] Didier Caucal. On Infinite Terms Having a Decidable Monadic Theory. In *Proc. of MFCS*, pages 165–176, 2002.

[Cau03] Didier Caucal. On Infinite Transition Graphs having a Decidable Monadic Theory. *Theoretical Computer Science*, 290(1):79–115, 2003.

[CC77] Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. of POPL*, pages 238–252, 1977.

[CDGL+11] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The Mastro system for ontology-based data access. *Semantic Web Journal*, 2011. To appear.

[CDLR09] Mika Cohen, Mads Dam, Alessio Lomuscio, and Francesco Russo. Abstraction in model checking multi-agent systems. In *Proc. of AAMAS (2)*, pages 945–952, 2009.

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CGJ+03] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[CGL94] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, MA, USA, 1999.

[CGRR10] P. Cangialosi, G. De Giacomo, R. De Masellis, and R. Rosati. Conjunctive Artifact-Centric Services. In *Proc. of ICSOC*, pages 318–333, 2010.

[Chu62] A. Church. Logic, Arithmetics, and Automata. In *Proc. of International Congress of Mathematicians*, 1962.

[Cou94] Bruno Courcelle. Monadic Second-Order Definable Graph Transductions: a Survey. *Theoretical Computer Science*, 126(1):53–75, 1994.

[CW96] Edmund M. Clarke and Jeannette M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[Dam94] Mads Dam. CTL* and ECTL* as Fragments of the Modal *mu*-Calculus. *Theoretical Computure Science*, 126(1):77–96, 1994.

[DDV11] Elio Damaggio, Alin Deutsch, and Victor Vianu. Artifact systems with data dependencies and arithmetic. In *Proc. of ICDT*, pages 66–77, 2011.

[DHPV09] Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic verification of data-centric business processes. In *Proc. of ICDT*, pages 252–267, 2009.

[DMS+05] Alin Deutsch, Monica Marcus, Liying Sui, Victor Vianu, and Dayou Zhou. A Verifier for Interactive, Data-Driven Web Applications. In *Proc. of SIGMOD Conference*, pages 539–550, 2005.

[DSV07] Alin Deutsch, Liying Sui, and Victor Vianu. Specification and verification of data-driven Web applications. *J. Comput. Syst. Sci.*, 73(3):442–474, 2007.

[DSVZ06] Alin Deutsch, Liying Sui, Victor Vianu, and Dayou Zhou. A system for specification and verification of interactive, data-driven web applications. In *Proc. of SIGMOD Conference*, pages 772–774, 2006.

[EC80] E. Allen Emerson and Edmund M. Clarke. Characterizing Correctness Properties of Parallel Programs Using Fixpoints. In *Proc. of ICALP*, pages 169–181, 1980.

[EH86] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[Eme96] E. Allen Emerson. Automated Temporal Reasoning about Reactive Systems. In Faron Moller and Graham Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *LNCS*, pages 41–101. Springer, 1996.

[FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. The MIT Press, 1995.

[FKMP05] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data Exchange: Semantics and Query Answering. *Theor. Comp. Sci.*, 336(1):89–124, 2005.

[FL02] Alain Finkel and Jérôme Leroux. How to Compose Presburger-Accelerations: Applications to Broadcast Protocols. In *Proc. of FSTTCS*, pages 145–156, 2002.

[GFPS10] Giuseppe De Giacomo, Paolo Felli, Fabio Patrizi, and Sebastian Sardiña. Two-Player Game Structures for Generalized Planning and Agent Composition. In *AAAI*, 2010.

[GMP09] Giuseppe De Giacomo, Riccardo De Masellis, and Fabio Patrizi. Composition of Partially Observable Services Exporting their Behaviour. In *Proc. of ICAPS*, 2009.

[GP09] Giuseppe De Giacomo and Fabio Patrizi. Automated Composition of Nondeterministic Stateful Services. In *Proc. of WS-FM (Revised Selected Papers)*, pages 147–160, 2009.

[GS84] F. Gcseg and M. Steinby. Tree Automata, 1984.

[GS97] Susanne Graf and Hassen Saïdi. Construction of Abstract State Graphs with PVS. In *Proc. of CAV*, pages 72–83, 1997.

[HR99] Michael R. A. Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems*. Cambridge University Press, 1999.

[JN00] Bengt Jonsson and Marcus Nilsson. Transitive Closures of Regular Relations for Verifying Infinite-State Systems. In *Proc. of TACAS*, pages 220–234, 2000.

[KLP04] Magdalena Kacprzak, Alessio Lomuscio, and Wojciech Penczek. From Bounded to Unbounded Model Checking for Temporal Epistemic Logic. *Fundamenta Informaticae*, 63(2-3):221–240, 2004.

[KM91] Hirofumi Katsuno and Alberto Mendelzon. On the Difference Between Updating a Knowledge Base and Revising It. In *Proc. of KR'91*, pages 387–394, 1991.

[KMM+01] Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. Symbolic model checking with rich assertional languages. *Theor. Comput. Sci.*, 256(1-2):93–112, 2001.

[Kol05] Phokion G. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *Proc. of PODS 2005*, pages 61–75, 2005.

[Koz83] Dexter Kozen. Results on the Propositional mu-Calculus. *Theoretical Computoter Science*, 27:333–354, 1983.

[Len02] Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. of PODS 2002*, pages 233–246, 2002.

[LPP70] David C. Luckham, David Michael Ritchie Park, and Mike Paterson. On Formalised Computer Programs. *J. of Computer and System Sciences*, 4(3):220–249, 1970.

[LQR09] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In *Proc. of CAV*, pages 682–688, 2009.

[LQS08a] Alessio Lomuscio, Hongyang Qu, and Monika Solanki. Towards Verifying Compliance in Agent-based Web Service Compositions. In *Proc. of AAMAS (1)*, pages 265–272, 2008.

[LQS08b] Alessio Lomuscio, Hongyang Qu, and Monika Solanki. Towards Verifying Contract Regulated Service Composition. In *Proc. of ICWS*, pages 254–261, 2008.

[LQSS07] Alessio Lomuscio, Hongyang Qu, Marek J. Sergot, and Monika Solanki. Verifying Temporal and Epistemic Properties of Web Service Compositions. In *Proc. of ICSOC*, pages 456–461, 2007.

[MP06] Angelo Montanari and Gabriele Puppis. Verification of Infinite State Systems. Lecture Notes for the 18th Summer School in Logic, Language and Information (ESSLLI'06), 2006.

[MS85] David E. Muller and Paul E. Schupp. The Theory of Ends, Pushdown Automata, and Second-Order Logic. *Theor. Comput. Sci.*, 37:51–75, 1985.

[Pnu81] Amir Pnueli. A Temporal Logic of Concurrent Programs. *Theoretical Computer Science*, 13:45–60, 1981.

[PPS06] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of Reactive(1) Designs. In *Proc. of VMCAI*, pages 364–380, 2006.

[PR89] Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *Proc. of POPL*, pages 179–190, 1989.

[PSZ10] Amir Pnueli, Yaniv Sa'ar, and Lenore D. Zuck. JTLV: A Framework for Developing Verification Algorithms. In *Proc. of CAV*, pages 171–174, 2010.

[Rab72] M.O. Rabin. *Automata on Infinite Objects and Church's Problem.* American Mathematical Society, Boston, MA, USA, 1972.

[Rei01] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* The MIT Press, 2001.

[RV10] Kristin Y. Rozier and Moshe Y. Vardi. LTL Satisfiability Checking. *International Journal on Software Tools for Technology Transfer*, 12(2):123–137, 2010.

[Saï00] Hassen Saïdi. Model Checking Guided Abstraction and Analysis. In *Proc. of SAS*, pages 377–396, 2000.

[SC85] A. Prasad Sistla and Edmund M. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3):733–749, 1985.

[Sem84] Alexei L. Semenov. Decidability of Monadic Theories. In *Proc. of MFCS*, pages 162–175, 1984.

[Spi03] Marc Spielmann. Verification of relational transducers for electronic commerce. *J. Comput. Syst. Sci.*, 66(1):40–65, 2003.

[vdMS99] Ron van der Meyden and Nikolay V. Shilov. Model Checking Knowledge and Time in Systems with Perfect Recall. In *Proc. of FSTTCS*, pages 432–445, 1999.

[Via09] Victor Vianu. Automatic verification of database-driven systems: a new frontier. In *Proc. of ICDT*, pages 1–13, 2009.

[Wal00] Igor Walukiewicz. Model Checking CTL Properties of Pushdown Systems. In *Proc. of FSTTCS*, pages 127–138, 2000.

[Wal02] Igor Walukiewicz. Monadic Second-Order Logic on Tree-like Structures. *Theoretical Computer Science*, 275(1-2):311–346, 2002.

[Wol86] P. Wolper. Expressing Interesting Properties of Programs in Propositional Temporal Logic. In *Proc. of POPL*, pages 184–193, 1986.

[Woo00] M. Wooldridge. Computationally Grounded Theories of Agency. In *Proc. of ICMAS*, pages 13–22. IEEE Press, 2000.

[Woo09] Michael J. Wooldridge. *An Introduction to MultiAgent Systems (2. ed.)*. Wiley, 2009.