Comparing the expressiveness of artifact-centric workflows

Diego Calvanese$^1$, Giuseppe De Giacomo$^2$, Rick Hull$^3$, Jianwen Su$^4$

$^1$ Free University of Bozen-Bolzano
$^2$ SAPIENZA University of Rome
$^3$ IBM T.J. Watson Research Center
$^4$ Univ. of California at Santa Barbara

INFINT 2009, Bertinoro, Italy
March 15–20, 2009
Motivations

We address the basic problem of comparing two artifact-based workflows wrt the executions that they can perform.

\( \sim \) **Dominance** between two workflows

Useful in different contexts:

- Update of WFs: Does the updated WF dominate the original one?
- Merge of two workflows: Does the merged WF dominate the two original WFs?
- WF optimization: The optimized WF should admit the same enactments as the original one.
Issues to address when comparing workflows

- How do we account for the fact that the services/tasks of the two workflows might be different.
- How does the performer come into play in the comparison?
- How do we take into account the choices that the performer can make?
- What kinds of artifacts do we allow?
- How do we take into account that the two workflows might operate on different artifacts?
- Do we want to take into account what happens during the execution?
Overview

1 Motivations
2 Workflow model
3 Dominance between workflows
4 Results for checking dominance
Basic assumptions

- We assume a first-order logic $\mathcal{L}$ with: equality, the usual logic symbols ($\land$, $\lor$, $\neg$, $\exists$, $\ldots$), constants, and possibly function and predicate symbols.

- We consider an $\mathcal{L}$-structure $\mathcal{S}$ with a nonempty universe $\mathcal{D}$. 
  \textit{Note}: in general, the universe is not finite $\leadsto$ We are infinite state.

- The structure $\mathcal{S}$ might be:
  - an arbitrary structure
  - a dense or discrete order ($<$)
  - Presburger arithmetic
  - a real (closed) field
Artifact-based workflows

- We follow a declarative approach to workflows.
- A workflow contains a set of services (or tasks) to be executed.
- (Business) rules establish the conditions for the execution of services (used to specify the lifecycle of the workflow).
- The data manipulated by the services is represented explicitly, through an artifact.

**Workflow schema**

Is a triple $\mathcal{W} = (\mathcal{A}, \mathcal{S}, \mathcal{R})$ where:

- $\mathcal{A}$ is an artifact schema, describing the artifacts of $\mathcal{W}$;
- $\mathcal{S}$ is a finite set of services;
- $\mathcal{R}$ is a finite set of rules.

We call the pair $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ a **pre-schema**.
Artifacts

- An artifact corresponds to a key business-relevant object.
- We take a simple view, where an artifact stores a fixed finite set of scalar values from some given domain.
- Each value is identified by an attribute, and we distinguish input and output attributes.
- Additionally, we may have attributes that store temporary values passed between services.

**Artifact schema**

Is a non-empty set $A$ of attribute names, partitioned into:

- a set $I_A$ of *input* attributes,
- a set $T_A$ of *temporary* attributes,
- a nonempty set $O_A$ of *output* attributes.

Attributes are single valued and scalar.
A snapshot of an artifact represents the status of the artifact at a given moment in time.

**Snapshot of an artifact schema $\mathcal{A}$**

Is a total function $\sigma : \mathcal{A} \rightarrow D \cup \{\bot\}$. (where $\bot$ means *undefined*)

- $\sigma$ is **initial** if
  - all input attributes of $\mathcal{A}$ are defined (i.e., $\neq \bot$), and
  - all temporary and all output attributes of $\mathcal{A}$ are undefined.

- $\sigma$ is **complete** if all output attributes of $\mathcal{A}$ are defined.
Services

The artifact gets modified by executing the services of the workflow.

**Service** for an artifact schema $\mathcal{A}$

Is a 4-tuple $S = (I_S, O_S, \delta_S, \xi_S)$, where

- $I_S \subseteq I_\mathcal{A} \cup T_\mathcal{A}$ are the input attributes of $S$.
- $O_S \subseteq T_\mathcal{A} \cup O_\mathcal{A}$ are the output attributes of $S$.
- $\delta_S$ is the pre-condition of $S$, i.e., an $\mathcal{L}$-formula over the input $I_S$, expressing the condition under which $S$ can be activated.
- $\xi_S$ is the post-condition of $S$, i.e., an $\mathcal{L}$-formula over the input $I_S$ and the output $O_S$, expressing the condition holding after the execution of $S$.

All attributes not in the output $O_S$ are not modified by $S$. This is expressed by the frame formula of $S$:

$$\Phi_S \equiv \bigwedge_{a \in \mathcal{A}\setminus O_S} (a = a').$$
The execution of a service typically changes the snapshot of the artifact.

**Execution of a service $S$**

Is a pair $(\sigma, \sigma')$ of snapshots of $A$ such that $(\sigma, \sigma') \models_S \delta_S \land \xi_S \land \Phi_S$.

- $\xi_S$ generally contains primed output attribute symbols, which denote the values of the output attributes after the execution of $S$.
- $(\sigma, \sigma') \models_S \varphi$ means that $\varphi$ is satisfied in $S$, when we consider $\sigma$ as an assignment to the unprimed attributes, and $\sigma'$ as an assignment to the primed attributes.

**Non-determinism** for executions means that a service $S$ may have two executions $(\sigma, \sigma')$ and $(\sigma, \sigma'')$, with $\sigma' \neq \sigma''$. 
Workflow enactments

An enactment is a sequence of service executions that, from an initial snapshot, may lead to a complete one (i.e., one where all outputs are defined).

**Enactment** of a pre-schema $\mathcal{P} = (\mathcal{A}, \mathcal{S})$

Is a sequence $E = \sigma_0, S_1, \sigma_1, \ldots, S_n, \sigma_n$ where:
- each $\sigma_i$ is a snapshot of $\mathcal{A}$, and each $S_i$ a service in $\mathcal{S}$,
- $\sigma_0$ is an initial snapshot (i.e., all input attributes are defined),
- each $(\sigma_{i-1}, \sigma_i)$ is an execution of service $S_i$.

The enactment is **complete** if $\sigma_n$ is complete.

We are interested in enactments that are supported by the rules of the workflow.
Rules of a workflow

The (business) rules specify the conditions under which a service may be executed.

**Rule** for a pre-schema \( \mathcal{P} = (\mathcal{A}, S) \)

Is an expression of the form \( (\text{if } \alpha \text{ allow } S) \), where

- \( \alpha \) is an \( \mathcal{L} \)-formula whose free variables are among the attribute names of \( \mathcal{A} \), and
- \( S \) is a service in \( S \).

**Enactment of a workflow schema** \( \mathcal{W} = (\mathcal{A}, S, \mathcal{R}) \)

Is an enactment \( \sigma_0, S_1, \sigma_1, \ldots, S_n, \sigma_n \) of \( (\mathcal{A}, S) \) such that, for each \( i \), there is a rule \( (\text{if } \alpha \text{ allow } S_i) \) in \( \mathcal{R} \) such that \( \sigma_{i-1} \models_S \alpha \).
The temporary attributes of a workflow are not visible to the outside of the workflow.

Hence, we are interested in the input-output behaviour of the enactments of the workflow.

We consider only the last written value for an output attribute.

**I/O-pair** of a complete enactment $E = \sigma_0, S_1, \sigma_1, \ldots, S_n, \sigma_n$

Is the pair $IO(E) = (\sigma_n|_{I_A}, \sigma_n|_{O_A})$.

*Note:* since the input attributes of the artifact are not changed by the services, we have that $\sigma_n|_{I_A} = \sigma_0|_{I_A}$.

$\sigma|_X$ denotes the restriction of the snapshot $\sigma$ to the attributes in $X$. 
Performers

A central role in the workflow is given to the performer(s) of services:

- The performers are often humans, but could also be software components.

- Various possibilities for how many performers we may have:
  - one performer for each service execution;
  - one performer for each service;
  - one performer for the whole workflow.

- When executing a service, the performer chooses the output values, among those that satisfy (under the given inputs) the conditions.

- In general, this may be done by exercising human judgement, and/or exploiting external information (which is not modeled explicitly) $\sim$ Executions may be non-deterministic.
Performance policies

A **performance policy** specifies conditions on the possible behaviors of the performers of the services of a workflow schema.

We may consider various significant cases for the performance policy:

- **Absolute**: the performers may use any execution of a service that satisfies its pre- and post-conditions (and the frame formula). Does not impose any restriction on the performers, hence captures the fact that a performer may use external non-modeled information in computing the output of a service.

- **Fixed-choice**: the computation done by a service $S$ must be a function of only the input attributes $I_S$ of $S$. The behaviour of the performer is completely determined by the inputs to the service.

- **Others**: computable, continuous, generic, ...
Performance policies (cont’d)

Formally, a performance policy $\pi$ for a workflow schema associates to each service $S$ a relation $\pi[S]$ over the values of the input and output attributes of $S$.

Let $X \subseteq A$. An **amap over** $X$ is a total function $\beta : X \rightarrow D \cup \{\bot\}$. 

*Note:*
- An amap over $A$ is just a snapshot of $A$.
- Hence, an amap over $X$ is the projection on $X$ of a snapshot.

Let $M[X]$ be the set of amaps over $X$.

**Performance policy** for a workflow schema $\mathcal{W} = (A, S, R)$

Is a function $\pi$:

- The domain of $\pi$ is the set $S$ of services.
- The value of $\pi$ on $S$, denoted $\pi[S]$, is a subset of $M[I_S] \times M[O_S]$ such that, if $(\mu, \nu) \in \pi[S]$, then $\mu \models_S \delta_S$ and $(\mu, \nu) \models_S \xi_S$. 

Compliance to a performance policy

Consider a workflow schema \( \mathcal{W} = (\mathcal{A}, \mathcal{S}, \mathcal{R}) \) and a performance policy \( \pi \).

- An execution \( (\sigma, \sigma') \) of a service \( S \) is **compliant** with \( \pi \) if
  \[
  (\sigma|_{I_S}, \sigma'|_{O_S}) \in \pi[S].
  \]

- An enactment \( E \) of \( \mathcal{W} \) is **compliant** with \( \pi \) if each execution in the enactment is compliant with \( \pi \).

- An I/O-pair is **realizable** under \( \pi \), if it is the I/O-pair of some enactment of \( \mathcal{W} \) compliant with \( \pi \).
Comparison of workflow schemas

We want to compare two workflow schemas $\mathcal{W}_1 = (A_1, S_1, R_1)$ and $\mathcal{W}_2 = (A_2, S_2, R_2)$.

- We compare only in terms of how values for the input attributes are mapped into values for the output attributes.

- Hence, we assume that $A_1$ and $A_2$ have the same input and output attributes (though they may differ in the temporary attributes). $\sim \mathcal{W}_1$ and $\mathcal{W}_2$ are **compatible**.

- We ignore the order in which the output attributes are written in one enactment versus the other enactment.
Dominance between workflows

Consider two compatible workflow schemas $\mathcal{W}_1$ and $\mathcal{W}_2$. Let $\Pi$ be a class of performance policies.

$\mathcal{W}_1$ is $\Pi$-dominated by $\mathcal{W}_2$, denoted $\mathcal{W}_1 \preceq_\Pi \mathcal{W}_2$, if:

for every performance policy $\pi_1 \in \Pi$ for $\mathcal{W}_1$

there exists a performance policy $\pi_2 \in \Pi$ for $\mathcal{W}_2$ s.t.

for every enactment $E_1$ of $\mathcal{W}_1$ compliant with $\pi_1$

there exists an enactment $E_2$ of $\mathcal{W}_2$ compliant with $\pi_2$ s.t.

$$IO(E_1) \subseteq IO(E_2).$$

Definition of $(\Pi, k)$-dominance, denoted $\mathcal{W}_1 \preceq_\Pi^k \mathcal{W}_2$, is similar, but we consider only enactments of length up to $k$. 


Different notions of dominance

We focus here on two classes of performance policies.

- $Abs$ denotes the class of all performance policies.
  - $\sim$ Absolute dominance, denoted $\mathcal{W}_1 \preceq_{Abs} \mathcal{W}_2$.
  - Absolute $k$-dominance, denoted $\mathcal{W}_1 \preceq^k_{Abs} \mathcal{W}_2$.

- A performance policy $\pi$ is fixed-choice if $\pi[S]$ is a function from $\mathcal{M}[I_S]$ to $\mathcal{M}[O_S]$ for each service $S$.
  - $FC$ denotes the class of all fixed-choice performance policies.
  - $\sim$ Fixed-choice dominance, denoted $\mathcal{W}_1 \preceq_{FC} \mathcal{W}_2$.
  - Fixed-choice $k$-dominance, denoted $\mathcal{W}_1 \preceq^k_{FC} \mathcal{W}_2$.
Deciding absolute dominance

- Since we allow for arbitrary policies, we have no restriction on the kind of computations that services can perform.

- Hence, to check whether $\mathcal{W}_1 \leq_{Abs}^{(k)} \mathcal{W}_2$, it suffices to compare “reachability” from inputs to outputs.

- Basic idea: for a workflow schema $\mathcal{W}$, we characterize the set of I/O pairs of $\mathcal{W}$ that are realizable (under some arbitrary performance policy).

- To do so, we build a FOL formula $\Psi_{\mathcal{W}}$ whose free variables are the input ($I_A$) and output ($O_A$) attributes of $A$.

- We check whether $\models_S \forall I_A \forall O_A (\Psi_{\mathcal{W}_1} \rightarrow \Psi_{\mathcal{W}_2})$. We get decidability in those cases where the FOL theory over the structure $S$ is decidable.
Approaches for characterizing the I/O-pairs

We have considered two approaches constructing $\Psi_W$:

- For bounded-length enactments: construct inductively the “cumulative postcondition”, projecting out at each step non-needed attributes.

- For arbitrary enactments: adopt the framework of constraint query languages [Kanellakis, Kuper, Revesz, JCSS’95], and construct a constraint Datalog program.
We consider a bound $k$ on the length of enactments.

1. We consider all sequences $\alpha_1 S_1 \alpha_2 S_2 \cdots \alpha_n S_n$ of length $\leq k$, where $\alpha_i$ is the condition of some rule (if $\alpha_i$ allow $S_i$).

2. For each $i$, we consider the various formulas involving $S_i$, while appropriately renaming variables:

$$
\begin{align*}
\alpha^p_i &= \alpha_i[a \to a^{i-1} \mid a \in A] \\
\delta^p_i &= \delta_{S_i}[a \to a^{i-1} \mid a \in A] \\
\xi^p_i &= \xi_{S_i}[a \to a^{i-1} \mid a \in I_{S_i}][a' \to a^i \mid a \in O_{S_i}] \\
\Phi^p_i &= \Phi_{S_i}[a \to a^{i-1} \mid a \in (A \setminus O_{S_i})][a' \to a^i \mid a \in (A - O_{S_i})]
\end{align*}
$$

3. We build inductively the “cumulative post-condition” $\hat{\xi}_i(\vec{a})$:

- $\hat{\xi}_0 = true$.
- $\hat{\xi}_i = \exists\{a^{i-1} \mid a \in A\}(\hat{\xi}^p_{i-1} \land \alpha^p_i \land \delta^p_i \land \xi^p_i \land \Phi^p_i)$

4. Let $\Psi^p_{\mathcal{W}} = (\exists\{a^n \mid a \in T_A\}(\hat{\xi}^p_n \land \bigwedge_{a \in O_A} a \neq \bot))[a^n \to a \mid a \in I_A \cup O_A]$

$$
\Psi_{\mathcal{W}} = \bigvee \text{sequences of length } \leq k
\Psi^p_{\mathcal{W}}
$$
Approach based on constraint Datalog program

1. Using the service pre- and post-conditions, and the rules specifications, we construct a constraint Datalog program $CDP_W$.

2. We evaluate $CDP_W$. [Kanellakis, Kuper, Revesz, JCSS’95] provides conditions under which the evaluation of this constraint Datalog program is guaranteed to terminate (also in the case of unbounded executions).

3. The evaluation of $CDP_W$ provides a FOL formula whose atoms are the $\mathcal{L}$-formulas accumulated during the evaluation.
Decidability results for absolute dominance

Theorem

$k$-absolute dominance is decidable when $\mathcal{L}$ is FOL with equality, and:

- $\mathcal{S} = (\mathbb{Q}, +, <)$, with the rationals as constants.
- $\mathcal{S} = (\mathbb{R}, +, <)$, with the rationals as constants.
- $\mathcal{S} = (\mathbb{Z}, +, <)$, with the integers as constants.
- $\mathcal{S} = (\mathbb{R}, +, \cdot)$, with the rationals as constants.

For the case of unbounded execution-length, we obtain decidability when the logic admits quantifier elimination.

Theorem

Absolute dominance is decidable when $\mathcal{L}$ is FOL with equality, and:

- $\mathcal{S} = (\mathbb{Q}, <)$, with the rationals as constants.
- $\mathcal{S} = (\mathbb{R}, <)$, with the rationals as constants.
- $\mathcal{S} = (\mathbb{Z}, <)$, with the integers as constants.
Theorem

Absolute dominance is undecidable when $\mathcal{L}$ is FOL with equality, and

- $S = (\mathbb{Q}, +)$, with the rationals as constants.
- $S = (\mathbb{R}, +)$, with the rationals as constants.

Idea:

- Use recursion to generate the integers, and to express multiplication by means of addition. Then check whether a polynomial equation in the integers has a solution.
- Alternatively, simulate a 2-counter machine.
For a fixed-choice policy $\pi$ for $\mathcal{W}$, we consider the set $IO_{FC}(\mathcal{W}, \pi)$ of all I/O-pairs of complete enactments of $\mathcal{W}$ compliant with $\pi$. We define the union of all such sets, for all fixed-choice policies $\pi$:

$$IO_{FC}(\mathcal{W}) = \bigcup_{\pi \in FC} IO_{FC}(\mathcal{W}, \pi)$$

**Lemma**

Checking emptiness of $IO_{FC}(\mathcal{W})$ is undecidable when $\mathcal{L}$ is a logic with equality over an infinite structure.

Proof based on reduction from PCP. We first use a linear order $<$ and then show that it can be eliminated.

**Theorem**

Let $\mathcal{L}$ be a logic with equality over an infinite structure. Then, checking fixed-choice dominance between workflow schemas over $\mathcal{L}$ is undecidable.
Deciding bounded fixed-choice dominance

We have no upper-bounds yet :-(

Conclusions

- First proposal for the comparison of the expressive power of artifact-centric workflows.

- Different notions of dominance, depending on the allowed performances policies.

- We have addressed the most simple case, where the artifact contains a fixed set of scalar values (of an unbounded domain).

- Preliminary results for the case of absolute dominance.

- The framework is open for several extensions:
  - set-valued attributes
  - artifacts contain relations
  - several artifact, possibly with constraints over them (e.g., an ontology)
Datalog formalization of I/O-pairs I

/* GENERATES ALL IO PAIRS OF COMPLETE COMPLIANT ENACTMENTS */
ioPairs(I,O1) :- initial(I,T,O),
transStar(I,T,O, I1,T1,O1),
complete(I1,T1,O1).

/* STATES WHEN A SNAPSHOT IS INITIAL */
initial(I,T,O) :- defined(I), //all vars in I are defined
undefined(T,O). //all vars in T,O are all undefined

/* STATES WHEN A SNAPSHOT IS COMPLETE */
complete(I,T,O) :- defined(O). //all vars in O are defined

/* COMPUTES THE REFLEXIVE TRANSITIVE CLOSURE OF TRANS */
transStar(I,T,O, I2,T2,O2) :- trans(I,T,O, I1,T1,O1),
transStar(I1,T1,O1, I2,T2,O2).
/* STATES THAT TRANSITION CAN BE MADE BY (AND ONLY BY) SERVICES */
trans(I,T,O, I1,T1,O1) :- transByServ_S1(I,T,O, I1,T1,O1).
...
trans(I,T,O, I1,T1,O1) :- transByServ_Sn(I,T,O, I1,T1,O1).

/* STATES THAT A TRANSITION IS MADE BY SERVICE S_i WHEN PRECONDITIONS AND RULE PRECONDITIONS HOLD, AND PRODUCE THE NEXT SNAPSHOT */
transByServ_Si(I,T,O, I1,T1,O1) :-
    rulesAllow_Si(I,T,O), //rules allow for service to apply
    delta_Si(I,T,O), //precondition on the subset must hold
    //constraints only I_Si of I,T.
    nextSnapshotByServ_Si(I,T,O, I1,T1,O1).

/* PRODUCES NEXT SNAPSHOT ON BASIS OF POSTCONDITION AND FRAME FORMULA FOR Si */
nextSnapshotByServ_Si(I,T,O, I1,T1,O1) :-
    chi_Si(I,T,O, I1,T1,O1), //constraints O_Si on the basis of I_Si
    //according to postcondition chi_Si
    phi_Si(I,T,O, I1,T1,O1). //frame formula
NOTE: chi_Si and phi_Si together constrain all variables I1,T1,O1
wrt I,T,O, either with effect chi_Si or with frame formula phi_Si

/* RULES ALLOWED FOR EACH SERVICE */
/* Let "if alpha_1 allow S_i"
   ...
   "if alpha_n allow S_n"
be all rules having S_i as consequent */
rulesAllow_S1(I,T,O) :- alpha_1(I,T,O).
   ...
rulesAllow_Sn(I,T,O) :- alpha_n(I,T,O).