Analyzing Service-Oriented Systems Using Their Data and Structure

Dragan Ivanović,[1] Manuel Carro,[1,2]
Manuel Hermenegildo[1,2]

[1]Universidad Politécnica de Madrid, [2]IMDEA Software Institute Madrid

S-Cube@ICSE 2012 – Zürich – June 5, 2012

Outline

- Analyze behavior of service (compositions) by taking into account complex control structures and impact of data.
  - Traditionally: stress on control structure.
    - E.g. Petri Nets, pi-calculus, STS, Reo.
    - But: loops/sub-workflows/compositionality/recursion: **non-trivial**!
  - Integrating the impact of data content / size:
    - On modeling / predicting { functional behavior / QoS properties }
- We present two of our approaches to:

1. Ensuring consistency in service compositions
2. Predicting SLA Violations

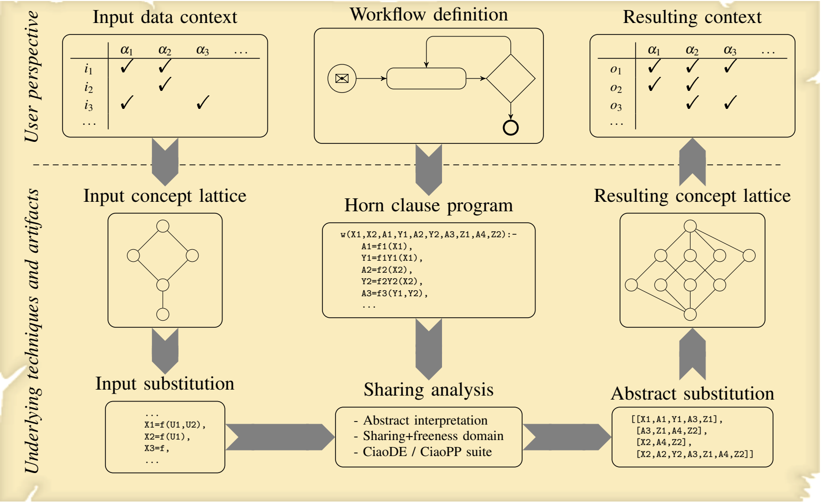
1 Consistency in Service Compositions

Data Attributes

- **User-defined attributes** can be used to characterize data
  - Domain-specific view – application dependent
  - E.g.: content, quality, privacy...
  - Possibly: a combination of views
  - Known for input data, implicit in control/data dependencies

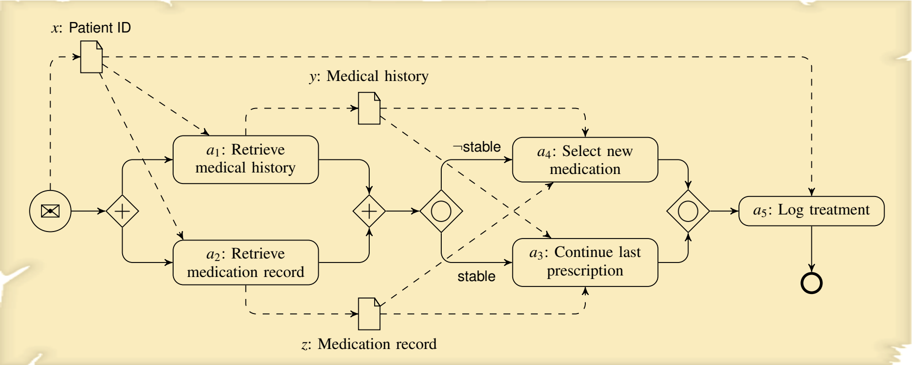
- **Challenge:** to infer user-defined attributes for data items and activities on different levels in an orchestration, automatically from:
  - known attributes of input data,
  - control structure, and
  - alertdata operations.

Approach



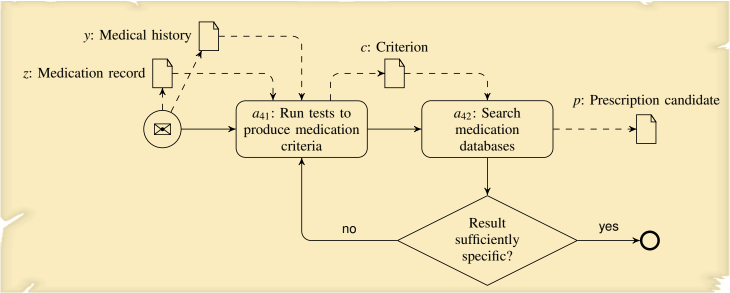
More info can be found in our previous work on automated attribute inference in complex service workflows [SCC-2011].

An Example Workflow



- An example showing medication prescription workflow.
- Written using BPMN (Business Process Modeling Notation).
  - A high-level (non-executable) description.

An Example Sub-Workflow



- Workflow implementing the component service a_4 in the main workflow.
- Involves sub-activities and additional data items.
- Includes looping based on data.

FCA Contexts

	Symptoms	Tests	Coverage
Medical history	✓	✓	
Medication record	✓		✓

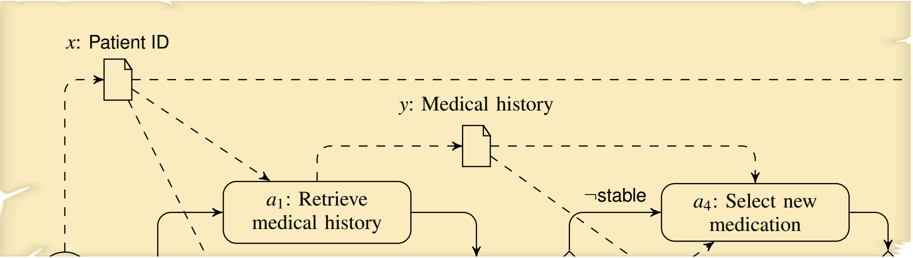
(a) Characteristics of medical databases.

	Name	Address	PIN	SSN
Passport	✓		✓	
National Id Card	✓	✓	✓	
Driving License	✓	✓		
Social Security Card	✓	✓		✓

(b) Types of identity documents.

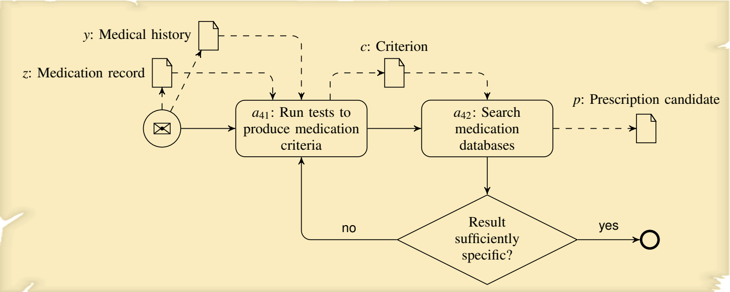
- Notions of context in Formal Concept Analysis (FCA): a Boolean relationship between objects and attributes.
  - E.g.: databases from which items *y* (Medical history) and *z* (Medication record) are retrieved use attributes *Symptoms*, *Tests* and *Coverage*.
  - If input *x* (Patient ID) is a passport, it has *Name* and *PIN*.
- Contexts can be converted into concept lattices.

Sharing in Orchestrations



- An activity inherits attributes of data it uses (reads).
  - The attributes may be inherited by data it writes.
  - It may introduce new attributes from its own sources.
- E.g.: a_1 reads x and the medical history database
  ⇒ a_1 and y share attributes *Name*, *PIN*, *Symptoms* and *Tests*.
- Sharing is transitive: e.g., a_4 shares all attributes of y.
- **Goal:** assign a minimal set of attributes to all activities and all intermediate / final data items in the orchestration.

Sharing and Complex Control



- Sharing analysis non-trivial in presence of complex control:
  - loops
  - branching (*if-then-else*)
  - recursion, non-determinism, etc.
- Solution: use approximation: minimal sharing superset conservative: no potential sharing excluded.

Sharing Analysis "Under the Hood"

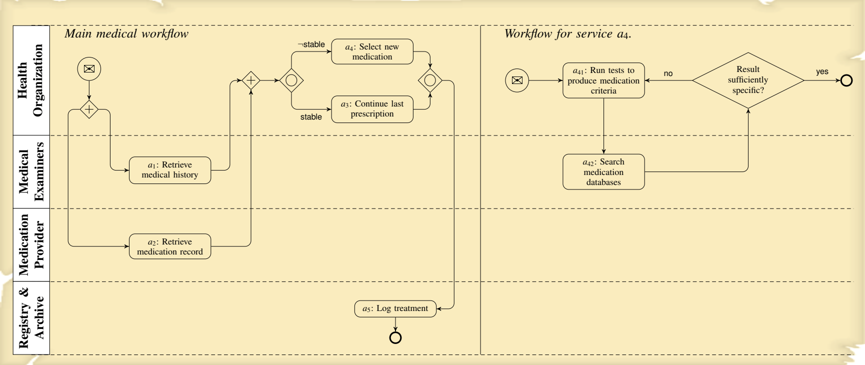
- Using sharing and freeness analysis for logic variables in Horn-clause programs.
  - based on abstract interpretation;
  - well-studied, powerful analysis tools (CiaoPP);
  - logic variables: placeholders for FOL terms ("sanitized pointers")
- Converting the workflow into a Horn-clause program.
  - mechanically;
  - keeping only the part of semantics relevant for sharing;
  - data items and activities → logic variables;
  - not mimicking full operational behavior
- The analysis works with and outputs abstract substitutions:
  - approximations that represent infinite families of sharing situations in a finite form;
  - can be set up from a context/lattice: input substitutions;
  - can be represented as a context/lattice: sharing results.

Resulting Context (From Sharing)

Item	Name	PIN	Symp.	Tests	Cover.
x	✓	✓			
d			✓	✓	
e			✓		✓
a_2, z	✓	✓	✓		✓
a_1, y, p, a_{42}, c	✓	✓	✓	✓	
a_3, a_4, a_{41}	✓	✓	✓	✓	✓
a_5	✓	✓			

- Attributes of input data preserved
  - **x**, **d**, **e** in the upper part
- Attributes of intermediate data & activities inferred from the lattice
  - For activities: attributes of the accessed data
  - Again: safe approximation – all potential attributes included

Information Flow Example



- Distributing execution of the workflow(s) across organizations
  - Composition fragments assigned to swim-lanes (partners)
  - Basis: protecting sensitive data
    - *Medical examiners* cannot see insurance coverage
    - *Medication providers* cannot see medical tests
    - *Registry* can see only the patient ID.

Applications

- Knowing the data attributes at design time can be used for:
  - Supporting fragmentation
    - *What parts can be enacted in a distributed fashion?*

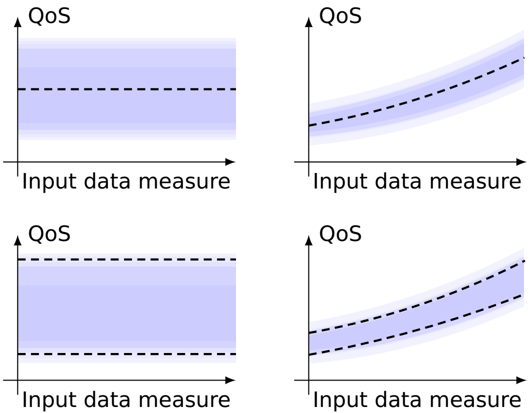
    e.g., based on the information flow.
  - Checking data compliance
    - *Is “sufficient” data passed to components?*

    e.g., can all activities be completed with all possible types of Patient ID?
  - Robust top-down development
    - *Refining specifications of workflow (sub-)components*

    e.g., iteratively decomposing “black box” composition components.

2 Predicting SLA Violations

Data-Sensitive QoS Bounds



Insensitive to Input Data | Sensitive to Input Data

Focus: Average Case

- Good for aggregate measures.
- Usually simpler to calculate.
- Not very informative for *individual running instances*.

Focus: Upper / Lower Bounds

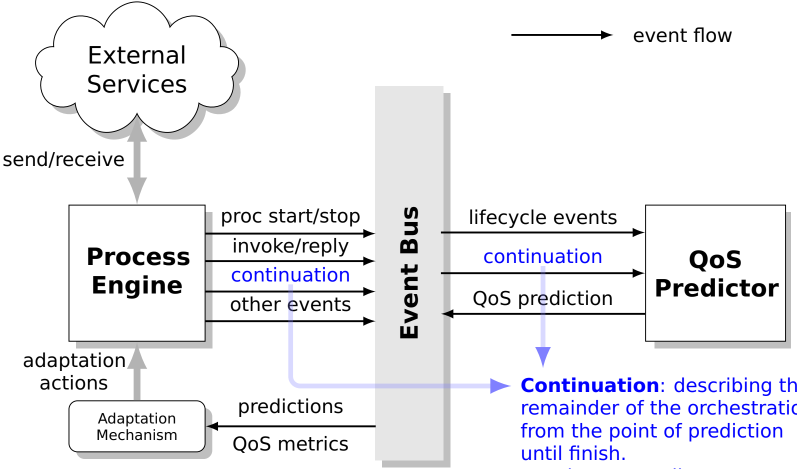
- Can be combined with the average case approach.
- More difficult to calculate.
- Useful for monitoring/ adapting *individual running instances*.

***General idea*: More information ⇒ more precision**

Motivation

1. Predicting **imminent SLA violations**:
   - Given knowledge on QoS metrics for component services.
   - Enabling us to abort / adapt ahead of time ⇒ prevention.
   - Inversely: certain SLA compliance ⇒ reuse of resources.
2. Predicting **potential SLA violations**:
   - Contingency planning for the case of failure.
   - Defining a range of adaptation actions.
3. Identifying **SLA succ/failure scenarios**: conditions and events that lead to SLA compliance/failure.
   - Exploring relationship between:
     - QoS metrics (overall and component services).
     - Structural parameters (branches, loops).
     - Data sent or received.

Overall Architecture



Continuation: describing the remainder of the orchestration from the point of prediction until finish.

- ⇒ *lower coupling*
- ⇒ *stateless implementation*

More info can be found in our previous work on constraint-based prediction of SLA violations [ICSOC-2011].

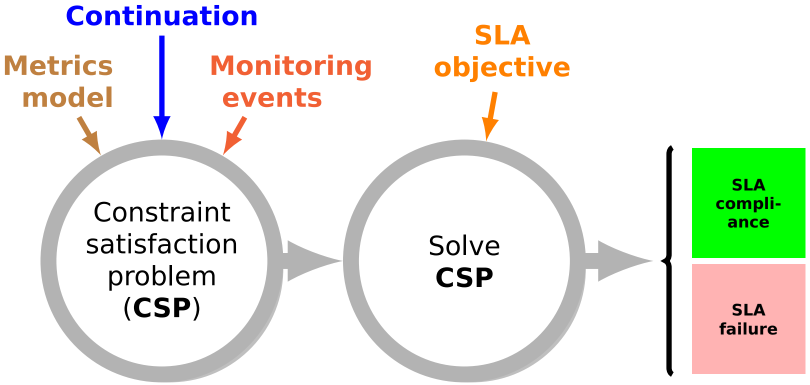
Continuations

- Use specific language for continuations.
  - Accepted by the predictor.
  - Used to derive constraint model.

- Obtaining continuation:
  - **By external observation:**
    - Needs orchestration definition, plus
    - orchestration / engine state, plus
    - lifecycle / execution events.

    May fall out of sync if information is incomplete or if the process is dynamically changed/adapted
  - **Directly from the execution engine:**
    - Always implicitly present in the interpreter state.
    - The engine may be “doctored” to provide it explicitly.
    - (Currently working on a prototype.)

Constraint-Based Prediction Steps



1. **Formulate a CSP** that models QoS for the executing orchestration instance.
2. **Solve the CSP** against the given SLA objective.
   - For two cases: **SLA compliance** and **SLA failure**.

Formulating CSP

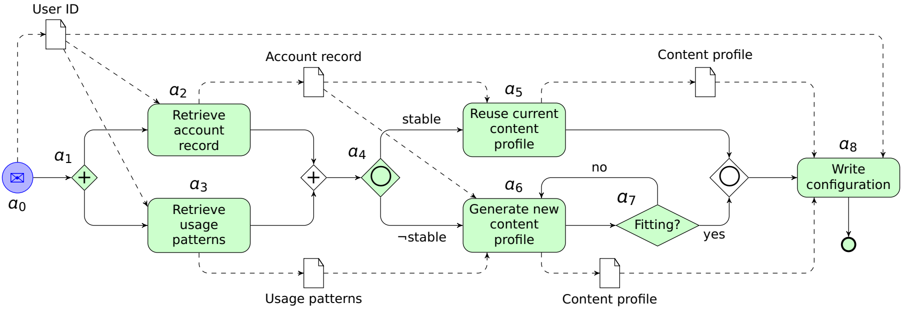
- **CSP built structurally** by decomposing the continuation into individual orchestration constructs:

sequences • parallel flows • service invocations • conditionals • loops

- QoS metrics of complex structures conservatively built from components' → logically sound if components' are sound.
- Metrics for the continuation = metrics for top-level construct.
- Can use **known run-time data** or **computational cost analysis** for services:
  - Infers upper and lower bound on # of iterations (k)
    - as functions of data
    - safe approximations
    - bounds coincide ⇒ exact k
  - Can be pre-computed statically or computed at run-time.

More info can be found in our previous work on predictive monitoring [MONA+2009] and data-aware QoS-driven adaptation [ICWS-2010] for service orchestrations.

Example: Prediction Inputs



Assumptions about components:

	Time bounds (ms) LB	UB
τ	0	10
a_2	500	800
a_3	200	500
a_5	100	400
a_6	200	600
a_8	100	300

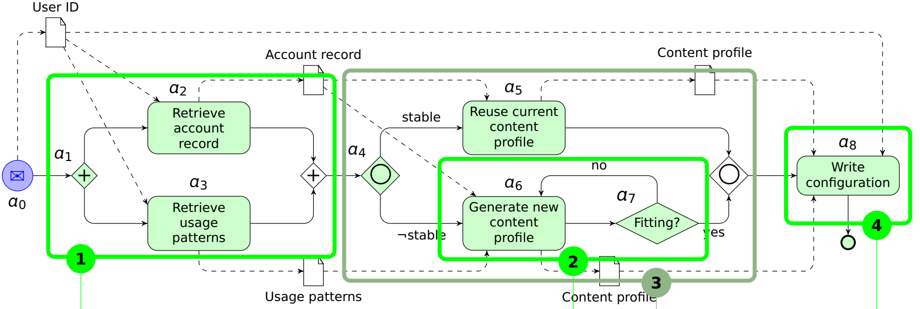
Metrics: execution time

SLA objective:

$$T_{max} = 1\,500\,\text{ms}$$

(from orchestration start)

Example (Cont.): Formulating CSP



$$500\,\text{ms} \leq T_{a_2} \leq 800\,\text{ms}$$
$$200\,\text{ms} \leq T_{a_3} \leq 500\,\text{ms}$$
$$\max(T_{a_2}, T_{a_3}) \leq T_{a_1} \leq T_{a_2} + T_{a_3}$$

$$200\,\text{ms} \leq T_{a_6} \leq 600\,\text{ms}$$
$$T' = (k+1) \times (\tau + T_{a_6})$$
$$k \in \mathbb{N}_0$$

$$100\,\text{ms} \leq T_{a_8} \leq 300\,\text{ms}$$

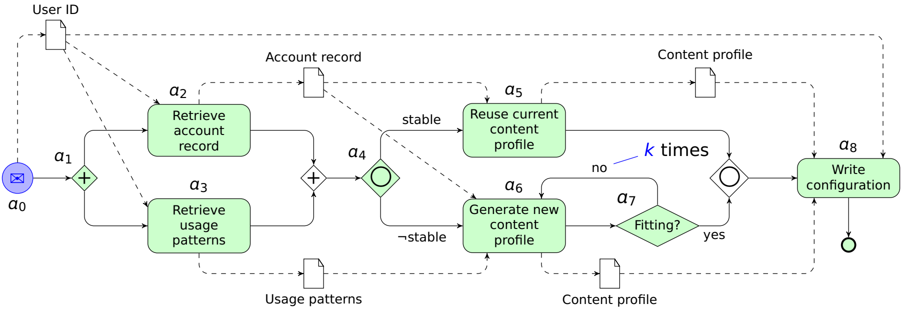
0

$$0 \leq \tau \leq 10\,\text{ms}$$

$$100\,\text{ms} \leq T_{a_5} \leq 400\,\text{ms}$$
$$(\text{stable} = 1 \wedge T_{a_4} = T_{a_5}) \vee (\text{stable} = 0 \wedge T_{a_4} = T')$$

$$T = T_{a_1} + T_{a_4} + T_{a_8} + 3 \times \tau$$

Example (Cont.): Solving CSP



$T \leq T_{max}$ when either:

- $stable = 1$, or
- $stable = 0$ and $k \leq 11$.

$T > T_{max}$ when:

- $stable = 0$ and $k \geq 3$

stable branch taken ⇒ SLA compliance ensured!
$k < 3$ at "yes" exit from a_7 ⇒ SLA compliance ensured!
$k \geq 12$ ⇒ imminent SLA failure!

(Prediction **at the orchestration start** – **becomes more precise later**.)

Evaluation

- Execution time of an industrial process: realistic data.
  - Ongoing work with colleagues from TUW and UniDuE.
  - 100 test runs, median execution time: 36 923 ms.
  - Continuous prediction (cca 160 times) for each instance.
  - Looking at first definite succ/fail prediction per instance.
  - T_{max} chosen to reflect failure rates between 0% and 100%.
- **High prediction accuracy** (94% to 100%) for different T_{max} (= % of correctly predicted cases)
- Prediction **timing**:
  - Able to predict SLA compliance early for reasonable failure rates.
  - SLA failures predicted between 5 000 ms and 9 000 ms before happening.
- Constraint-based prediction proven **very efficient**:
  - 295 to 490 ms to run 160 predictions per instance.
  - $\approx 1 - 2\%$ of instance execution time.

Outlook of Future Work

- **Sharing-based analysis** allows mathematical (object-attribute/lattice) treatment of data dependencies and properties.
  - Extend towards minimal sharing and adaptation constraints.
  - Automate derivation of Horn-Clause programs from executable specification (BPEL, XPDL, Yawl, etc.)
  - Extend to include stateful conversations.
- **Constraint-based QoS prediction** is a efficient, robust and accurate run-time technique for service orchestrations.x
  - Continue with experimental / real life evaluation.
  - Interfacing with various process engines.
  - Explore in depth the effects of inaccurate / imprecise information about component service QoS.
  - Enrich the model to cope with imprecision.

Analyzing Service-Oriented Systems Using Their Data and Structure

Dragan Ivanović,[1] Manuel Carro,[1,2]
Manuel Hermenegildo[1,2]

[1]Universidad Politécnica de Madrid, [2]IMDEA Software Institute Madrid

S-Cube@ICSE 2012 – Zürich – June 5, 2012