Integrated Modeling and Verification of Processes and Data
Exploiting DCDSs: models, methods, concrete systems

Diego Calvanese, Marco Montali

Research Centre for Knowledge and Data (KRDB)
Free University of Bozen-Bolzano, Italy

15th International Conference on Business Process Management
Barcelona, Spain – 12 September 2017
The story so far, with main references

- The need of **combining (business) processes and data**.
  [Calvanese, De Giacomo, and Montali 2013]

- A pristine formalism for data-aware business processes: **DCDS**.

- Suitable **verification logics** for data-aware processes.

- Corresponding **characterization theorems**.
  [Calvanese, De Giacomo, Montali, and Patrizi 2017]

- A **decidability map**, with an unexpected dichotomy between $\mu L_A$ and LTL-FO$_A$.

How to check/ensure state boundedness?

**Theorem**

Checking whether a DCDS is **state-/run-bounded** is:

- **Decidable** for a **given** bound.
- **Undecidable** for an **unknown** bound.

**Three possible strategies:**

- Single out **classes of DCDSs** for which checking state-/run-boundedness is decidable.
- Identify **sufficient syntactic conditions** that are decidable to check, and that guarantee state-/run-boundedness
  - cf. syntactic conditions for chase termination in data exchange.
- Devise **modeling methodologies** that guarantee state boundedness.
How to check/ensure state boundedness?

Theorem

**Checking** whether a DCDS is **state-/run-bounded** is:
- **Decidable** for a **given** bound.
- **Undecidable** for an **unknown** bound.

Three possible strategies:
- Single out **classes of DCDSs** for which checking state-/run-boundedness is decidable.
- Identify **sufficient syntactic conditions** that are decidable to check, and that guarantee state-/run-boundedness
  - cf. syntactic conditions for chase termination in data exchange.
- Devise **modeling methodologies** that guarantee state boundedness.
DCDSs with decidable state-boundedness

Fact

DCDSs using only unary relations correspond to variants of Petri nets.
- The specific variant depends on the features used in the DCDS.

Note: State-boundedness relates to boundedness in Petri nets.

Petri nets with name management

Decidable boundedness.
[Rosa-Velardo and Frutos-Escrig 2011]

Reset-Transfer Nets

Undecidable boundedness.
[Dufourd, Jancar, and Schnoebelen 1999]

Translation to DCDSs and $\mu\mathcal{L}_P$ verification.

“Lossy” correspondence with DCDSs.

[Rosa-Velardo and Frutos-Escrig 2011]

[Montali and Rivkin 2016]

[Bagheri Hariri, Calvanese, Deutsch, et al. 2014]
DCDSs with decidable state-boundedness

**Fact**

DCDSs using only unary relations correspond to variants of Petri nets.
- The specific variant depends on the features used in the DCDS.

*Note:* State-boundedness relate to boundedness in Petri nets.

**Petri nets with name management**

*Decidable* boundedness.

[Rosa-Velardo and Frutos-Escrig 2011]

**Reset-Transfer Nets**

*Undecidable* boundedness.

[Dufourd, Jancar, and Schnoebelen 1999]

[Montali and Rivkin 2016]

Translation to DCDSs and $\mu\mathcal{L}_P$ verification.

[Bagheri Hariri, Calvanese, Deutsch, et al. 2014]

“Lossy” correspondence with DCDSs.
The story so far

State-boundedness

Boundedness and resources

Unbounded systems

Concrete systems

DCDSs with decidable state-boundedness

Fact

DCDSs using only unary relations correspond to variants of Petri nets.

- The specific variant depends on the features used in the DCDS.

Note: State-boundedness relate to boundedness in Petri nets.

Petri nets with name management

Decidable boundedness.

[Rosa-Velardo and Frutos-Escrig 2011]

[Montali and Rivkin 2016]

Translation to DCDSs and \( \mu L_P \) verification.

Reset-Transfer Nets

Undecidable boundedness.

[Dufourd, Jancar, and Schnoebelen 1999]

[Bagheri Hariri, Calvanese, Deutsch, et al. 2014]

“Lossy” correspondence with DCDSs.
Attacking state-boundedness

The class of DCDSs with decidable state-boundedness very restrictive

These variants of Petri nets corresponds to DCDSs with only unary relations, limited use of negation, no or limited joins, ...

How to check/guarantee that a DCDS is state-bounded?

Sufficient, syntactic conditions:

- Extract a data flow graph from the DCDS.
- Check sources of unboundedness through this graph.


State-boundedness by design:

Design methods for state-bounded DCDSs. In [Solomakhin et al. 2013]:

- Processes are bound to evolving business objects (artifacts).
- Each business object manipulate boundedly many data.
- (New) business objects pick their names from a fixed pool of ids.

Attacking state-boundedness

The class of DCDSs with decidable state-boundedness very restrictive

These variants of Petri nets corresponds to DCDSs with only unary relations, limited use of negation, no or limited joins, ...

How to check/guarantee that a DCDS is state-bounded?

Sufficient, syntactic conditions:

- Extract a data flow graph from the DCDS.
- Check sources of unboundedness through this graph.


State-boundedness by design:

Design methods for state-bounded DCDSs. In [Solomakhin et al. 2013]:

- Processes are bound to evolving business objects (artifacts).
- Each business object manipulate boundedly many data.
- (New) business objects pick their names from a fixed pool of ids.

Attacking state-boundedness

The class of DCDSs with decidable state-boundedness very restrictive

These variants of Petri nets corresponds to DCDSs with only unary relations, limited use of negation, no or limited joins, ... 

How to check/guarantee that a DCDS is state-bounded?

Sufficient, syntactic conditions:
- Extract a data flow graph from the DCDS.
- Check sources of unboundedness through this graph.


State-boundedness by design:

Design methods for state-bounded DCDSs. In [Solomakhin et al. 2013]:
- Processes are bound to evolving business objects (artifacts).
- Each business object manipulate boundedly many data.
- (New) business objects pick their names from a fixed pool of ids.

### State-boundedness in concrete process modeling languages

#### Classical BPM languages/suites

- Central notion of **case** representing a process instance.
- Each case carries its own **case data**, in isolation to the other cases (e.g., order details, customer address, ...).
- Cases interact by accessing a central, **persistent data storage**.

#### Artifact-centric approaches:

- Central notion of **business object** gluing data and behaviour together.
- **All data** relevant to a business object are attached to it.
- Processes may query **multiple business objects** at once, to determine the possible next steps.

#### External and internal stakeholders...

- New cases/business objects are created upon **events** issued by external stakeholders (e.g., new order request).
- But then they are bound to **internal resources**, responsible for progressing the corresponding process instances.
# State-boundedness in concrete process modeling languages

## Classical BPM languages/suites

- Central notion of **case** representing a process instance.
- Each case carries its own **case data**, in isolation to the other cases (e.g., order details, customer address, ...).
- Cases interact by accessing a central, **persistent data storage**.

## Artifact-centric approaches:

- Central notion of **business object** gluing data and behaviour together.
- **All data** relevant to a business object are attached to it.
- Processes may query **multiple business objects** at once, to determine the possible next steps.

## External and internal stakeholders...

- **New cases/business objects** are created upon **events** issued by **external stakeholders** (e.g., new order request).
- But then they are bound to **internal resources**, responsible for progressing the corresponding process instances.
RIAW-nets [Montali and Rivkin 2016]

RIAW-nets = $\nu$-PNs + workflow nets

- Emitter transition generating a new process id when fired.
- Control-flow name matching to selectively spawn/synch tokens using their id.
- Resource places to bound the number of simultaneously coexisting active process instances! (but unboundedly many over time).

Decidability of model checking via translation to state-bounded DCDSs.
Data isolation and case unboundedness

What if the number of simultaneously active cases cannot be bounded?

In [Montali and Calvanese 2016; Calvanese, Montali, et al. 2014], we show that **decidability** of model checking can be retained, if the system obeys to:

- **relative boundedness** (each case manipulates boundedly many data);
- **data isolation** (cases interact very weakly).

### Modeling guidelines to guarantee data isolation and relative boundedness:

1. Queries must be navigational (no arbitrary access to relations).
2. 1-to-many relations require a number restriction on the “many” side.
3. Each case cannot create a chain of tuples of unbounded length.
4. Cases can share tuples only in a controlled way (no construction of chains).
Data isolation and case unboundedness

What if the number of simultaneously active cases cannot be bounded?

In [Montali and Calvanese 2016; Calvanese, Montali, et al. 2014], we show that **decidability** of model checking can be retained, if the system obeys to:

- relative boundedness (each case manipulates boundedly many data);
- data isolation (cases interact very weakly).

<table>
<thead>
<tr>
<th>State</th>
<th>Group</th>
<th>MarryM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>group</td>
<td>state</td>
<td>id</td>
</tr>
<tr>
<td>12</td>
<td>out</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>in</td>
<td>4</td>
</tr>
<tr>
<td>431</td>
<td>running</td>
<td>431</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>76</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>98</td>
</tr>
<tr>
<td>id</td>
<td>combatLevel</td>
<td>group</td>
</tr>
<tr>
<td>76</td>
<td>pro</td>
<td>null</td>
</tr>
<tr>
<td>19</td>
<td>basic</td>
<td>4</td>
</tr>
<tr>
<td>56</td>
<td>ok</td>
<td>431</td>
</tr>
<tr>
<td>3</td>
<td>basic</td>
<td>431</td>
</tr>
<tr>
<td>98</td>
<td>ok</td>
<td>431</td>
</tr>
</tbody>
</table>

**Modeling guidelines** to guarantee data isolation and relative boundedness:

1. Queries must be navigational (no arbitrary access to relations).
2. 1-to-many relations require a number restriction on the “many” side.
3. Each case cannot create a chain of tuples of unbounded length.
4. Cases can share tuples only in a controlled way (no construction of chains).
What if the number of simultaneously active cases cannot be bounded?

In [Montali and Calvanese 2016; Calvanese, Montali, et al. 2014], we show that **decidability** of model checking can be retained, if the system obeys to:

- **relative boundedness** (each case manipulates boundedly many data);
- **data isolation** (cases interact very weakly).

**Modeling guidelines** to guarantee data isolation and relative boundedness:

1. Queries must be navigational (no arbitrary access to relations).
2. 1-to-many relations require a number restriction on the “many” side.
3. Each case cannot create a chain of tuples of unbounded length.
4. Cases can share tuples only in a controlled way (no construction of chains).
Data isolation and case unboundedness

What if the number of simultaneously active cases cannot be bounded?

In [Montali and Calvanese 2016; Calvanese, Montali, et al. 2014], we show that **decidability** of model checking can be retained, if the system obeys to:

- **relative boundedness** (each case manipulates boundedly many data);
- **data isolation** (cases interact very weakly).

### Modeling guidelines to guarantee data isolation and relative boundedness:

1. Queries must be navigational (no arbitrary access to relations).
2. 1-to-many relations require a number restriction on the “many” side.
3. Each case cannot create a chain of tuples of unbounded length.
4. Cases can share tuples only in a controlled way (no construction of chains).
Beyond State-Boundedness

**Question**

Are there classes of DCDSs that are *unbounded*, but still *amenable to verification*?

Key result in [Abdulla et al. 2016].

**Recency-bounded data-aware processes**

Unbounded DB, but only the latest inserted/accessed values can bound to parameters.

**Verification via under-approximation**

*Decidability* by focusing only on runs that are $k$-recency-bounded for an explicitly given key.

**Open problem**

Investigate the relationships between all such results and those where the initial DB is *not fixed*, and verification is studied *for every possible* initial DB.
Beyond State-Boundedness

**Question**

Are there classes of DCDSs that are **unbounded**, but still **amenable to verification**?

Key result in [Abdulla et al. 2016].

**Recency-bounded data-aware processes**

Unbounded DB, but only the latest inserted/accessed values can bound to parameters.

Verification via under-approximation

**Decidability** by focusing only on runs that are k-recency-bounded for an explicitly given key.

**Open problem**

Investigate the relationships between all such results and those where the initial DB is **not fixed**, and verification is studied **for every possible** initial DB.
Beyond State-Boundedness

Question
Are there classes of DCDSs that are **unbounded**, but still **amenable to verification**?

Key result in [Abdulla et al. 2016].

**Recency-bounded** data-aware processes
Unbounded DB, but only the latest inserted/accessed values can bound to parameters.

Verification via under-approximation
**Decidability** by focusing only on runs that are k-recency-bounded for an explicitly given key.

Open problem
Investigate the relationships between all such results and those where the initial DB is **not fixed**, and verification is studied **for every possible** initial DB.
Incorporation of datatypes

Databases have **datatypes**

Numeric domains, domain-specific predicates, arithmetic.
- Many coordination algorithms and auctions require dense orders.
- Processes with costs and payment policies require integers and arithmetic.

Dense orders combine well with state-boundedness

Data-aware, state-bounded distributed systems with reals [Calvanese, Delzanno, and Montali 2015]:
- **OK** to include **dense linear orders**: minor extension to the standard DCDS abstraction technique. Intuition...
  - Rigid $>$ relation over the entire domain $\rightarrow$ Non-rigid $GreaterThan$ relation over active domain elements.
- **No hope** to include the **successor** relation (or integers):
  2 data slots are sufficient to encode two counters.

Discrete orders and arithmetic combine well with run-boundedness

Ongoing work...
Relational multiagent systems and commitments

Relational MAS [Montali, Calvanese, and De Giacomo 2014]

- Agents have names and hold/manipulate local, state-bounded DBs.
- Agents exchange data using their names for addressing.
- An institutional agent manages agent creation and deletion.
  - Due to state-boundedness: unboundedly many agents can dynamically enter into the system, but at each moment only boundedly many are active.

Relational commitments
In the same work: first proposal for modeling and verifying interaction protocols based on relational commitments, i.e., commitments with data payload and multiple instances.
daphne: implementing DCDSs with relational technology

Native **modeling and execution** of DCDSs using relational DBMSs:

- **SQL-like syntax** for DCDSs with datatypes.
- Automated translation into **relational DBMSs**, as (temporal) **tables**, **constraints**, and **stored procedures**.
- **Java APIs** to support enactment and integration with concrete **services**.

Native **explicit model checking** of DCDSs using relational DBMSs:

- **Same model for execution and verification!**
- **Special tables** for storing the RTS induced by a DCDSs.
- **Factoring of tables** into **temporal** and **atemporal** parts.
- **Computation** of **isomorphic type and value recycling** in services.
- **Java APIs** for RTS construction and search.
Can we cook with all ingredients?

**“REAL” PROCESS**

- Explicit control-flow
- Local, case data
- Global, persistent data
- Queries/updates on the persistent data
- External inputs
- Internal generation of fresh IDs
BAUML: artifact-centric processes with UML

**BAUML approach**

- **Business objects**, states, associations and attributes: UML class diagrams.
- Business object lifecycle: UML statechart diagram.
- Complex event triggering a lifecycle transition: UML activity diagram.
  - Tasks modeled as OCL operation contracts.

In [Calvanese, Montali, et al. 2014]: **methodology to guarantee decidability of model checking** (see before). Estanol PhD thesis: BAUML to DCDS!
raw-sys: marrying workflow nets and databases

Data-aware processes using **well-known formalisms**:

- **Data**: global and local relational databases.
- **Process control-flow**: workflow nets, enriched with:
  - **Guards** (queries over the DBs).
  - **STRIPS-like actions** with external inputs from an infinite domain, invoked upon firing net transitions.

**raw-sys model** [De Masellis et al. 2017]:

**raw-sys verification** [De Masellis et al. 2017]:

- Map of **(un)decidability**, exploiting **translation to DCDSs**.
- **Encoding into planning** systems to handle reachability problems.
db-nets: marrying colored Petri nets and databases

db-net model [Montali and Rivkin 2017], three layers:

1. **Persistence:** relational database with constraints.
2. **Data logic:** queries and actions over the persistence layer.
3. **Control:** colored Petri net with \( \nu \)-variables, enriched with **view places** and **transition-action bindings** to inspect/update the persistence layer.

**Note:** Natural formalization of contemporary process modeling suites!
db-nets: marrying colored Petri nets and databases

db-nets execution, simulation, verification [Montali and Rivkin 2017]:

- Foundational results thanks to translation to DCDSs.
- Ongoing implementation effort inside www.cpntools.org.
OCBC: declarative data+process integrated model

**OCBC model** [Artale et al. 2017], three components:

1. **Data model**: UML class diagram.
2. **Tasks**: units of work, referencing classes in the data model. Each task instance comes with objects belonging to such classes.
3. **Behavioral constraints**: declarative patterns equipped with coreference relations pointing to the data model. They constrain when tasks can be executed, and which data objects they should carry.

Naturally captures many-to-many processes with no single notion of case!
Acknowledgements

Thanks to the many people who contributed interesting ideas, suggestions, discussions, and collaborated to the presented results.

Giuseppe De Giacomo
Fabio Patrizi

Babak Bagheri Hariri
Riccardo De Masellis
Alin Deutsch
Marlon Dumas
Paolo Felli
Rick Hull
Maurizio Lenzerini
Alessio Lomuscio
Andy Rivkin
Ario Santoso
Thank you for your attention!


References


References IV


