Declarative Flexible Workflows for Trustworthy Pervasive Healthcare Services

Talk at Microsoft Research India, Bangalore
29 July, 2010

Raghava Rao Mukkamala
PhD student,
Advisor: Thomas T Hildebrandt
Programming, logic and Semantics Group
IT University of Copenhagen, Denmark
Overview

- PhD thesis Overview
- Problem statement and research questions
- Research Methodology and Approach
- Results up to date
- Formal Process model Dynamic condition response structures (DCRS)
- Model checking and verification of DCRS processes
- Road map for next 18 months
- Expected Contribution
PhD Thesis Overview

• Part of Trustworthy Pervasive Healthcare Services (TrustCare) project.

• Timeline:

- Thesis proposal defense in September at the university.

- Thesis Submission

03/2009 09/2010 02/2012
Research Questions

Overall Goal: To develop formal foundations for trustworthy and declarative flexible workflows with a key focus on the health care sector.

1. What are the formal semantical models suitable for describing flexible workflow processes for health care and other dynamic services?
   Formal workflow language based on declarative modeling primitives.

2. How should one describe interfaces, contracts and interactions for declarative, quantitative and dynamic workflows?
   Extensions to Session types/End-point projections for declarative and quantitative contracts.

3. What are the suitable model checking and verification techniques for enhancing trustworthiness of declarative, quantitative and dynamic workflows?
   Model checking and verification techniques for declarative processes.
Prior and Related Work

• Declare tool and DecSerFlow

• Event Calculus
  • Nihan Kesim Cicekli and Ilyas Cicekli: Formalizing the specification and execution of workflows using the event calculus. (2006)

• Session Types and Choreographies (WS-CDL)
  • Marco Carbone, Kohei Honda, and Nobuko Yoshida: Structured communication-centered programming for web services

• Prior work:
Research approach and methodology

TrustCare Project Methodology

Goal: To provide trustworthy it-support for pervasive health care service

- Development of Resultmaker Online Consultant
- Experimental research on user-interfaces for pervasive healthcare centered on activity based computing
- Domain knowledge on user-interfaces for pervasive healthcare
- Domain knowledge on workflow management and clinical guidelines
- Research in type theory and logical frameworks
- Advanced general solutions
- Domain-specific challenges for interface specification, dynamic changes, composition and decomposition
- Research in Domain-specific Languages and formal models

Fundamental typed models for concurrent interacting processes

Thomas Hildebrandt, [Raghava Rao Mukkamala]

Process IT (Interest Group)

BPM and Workflow Vendors

- Resultmaker A/S
- Jakob Bardram

Pervasive interfaces in hospital environment + activity based computing

Techniques for reasoning about types + communication models for interacting workflows (multi party session types)

Proof carrying code for interacting concurrent models + encoding Robin Milner's Bigraphs into CELF/TWELF

Models for distributed programming by contracts based game theory

Fritz Henglein, Carsten Schüermann, Andrzej Filinski
Motivation for Declarative Modeling

Traditional workflow example in BPMN

- Specification
  - 3 tasks: bless, curse and pray.
  - Rule: if you curse someone, then you MUST pray afterwards.

BPMN Symbols

Motivation for Declarative Modeling

Traditional workflow example in BPMN

• Specification
  ➡ 3 tasks: bless, curse and pray.
  ➡ Rule: if you curse someone, then you MUST pray afterwards.

[BPMN Symbols]

Motivation for Declarative Modeling

Traditional workflow example in BPMN

• Specification
  ➡ 3 tasks: bless, curse and pray.
  ➡ Rule: if you curse someone, then you MUST pray afterwards.

BPMN: Business Process Modeling Notation
Motivation for Declarative Modeling

Traditional workflow example in BPMN

- Specification
  - 3 tasks: bless, curse and pray.
  - Rule: if you curse someone, then you MUST pray afterwards.

BPMN Symbols

[bless]
Motivation for Declarative Modeling

Traditional workflow example in BPMN

• Specification
  ➡ 3 tasks: bless, curse and pray.
  ➡ Rule: if you curse someone, then you MUST pray afterwards.

BPMN Symbols

[bless]   [pray]

Motivation for Declarative Modeling

Traditional workflow example in BPMN

- Specification
  - 3 tasks: bless, curse and pray.
  - Rule: if you curse someone, then you MUST pray afterwards.

BPMN Symbols

1 BPMN: Business Process Modeling Notation
Motivation for Declarative Modeling

Traditional workflow example in BPMN\(^1\)

- **Specification**
  - 3 tasks: bless, curse and pray.
  - Rule: if you curse someone, then you MUST pray afterwards.

---

\(^1\) BPMN: Business Process Modeling Notation
Motivation for Declarative Modeling

Traditional workflow example in BPMN\(^1\)

• Specification
  ➡ 3 tasks: bless, curse and pray.
  ➡ Rule: if you curse someone, then you MUST pray afterwards.

BPMN Symbols

\(^1\) BPMN: Business Process Modeling Notation
Motivation for Declarative Modeling

Traditional workflow example in BPMN

- Specification
  - 3 tasks: bless, curse and pray.
  - Rule: if you curse someone, then you MUST pray afterwards.

BPMN Symbols

[bless]  [pray]  [curse, pray]  [bless], [pray], [curse, pray]

Motivation for Declarative Modeling

Traditional workflow example in BPMN¹

• Specification
  ➡ 3 tasks: bless, curse and pray.
  ➡ Rule: if you curse someone, then you MUST pray afterwards.

Control flow is explicit and hence we have to think ahead about all the possible computations we want to support.
Motivation for Declarative Modeling

Constraint-based Approach

• Specification
  ➡ 3 tasks: bless, curse and pray.
  ➡ Rule: if you curse someone, then you MUST pray afterwards.

• Tasks can be executed
  ✓ any number of times
  ✓ in any random order
  unless they are prevented by constraints

• Accepting run: any execution ending with accepting state, where all constraints are satisfied

Accepting runs:
[bless, bless]
[bless, bless, curse, pray]
[curse, curse, pray,]
[curse, curse, pray, bless, bless]

Non-accepting runs:
[pray, curse]
[bless, curse, pray, curse, bless]
Imperative versus Declarative models

**Imperative Models**
- Control flow is explicit hence over-specification, so less flexible
- Adding a new constraint requires to make a new model
- Focus is on *how* a set of tasks will be performed
- Very successful in sectors where strictly procedural execution is needed!

**Declarative Models**
- Control flow is implicit hence under-specification, so more flexible
- Specify constrains to forbid the unwanted behavior
- Focus is on *what* should be done instead of *how*
- Difficult to perceive
  - where to start and where to end
  - what are next possible actions

---

Dynamic Condition Response Structures (DCRS) - Overview

- DCRS is formal process model for specification and execution of flexible workflows/business processes.
- Using declarative modeling approach
- Based on Event Structures\textsuperscript{3}, a minimal, and declarative model for concurrent processes.
- DCRS also has a graphical notation which closely tied to the formal model.
- Execution semantics
  - For finite runs, mapped to labeled transition system
  - for infinite runs, mapped to generalized Büchi Automaton

Event Structures

Definition
A labeled prime event structure (ES) over an alphabet Act is a 4-tuple \((E, \leq, \#, I)\) where

- \(E\) is a (possibly infinite) set of events
- \(\leq \subseteq E \times E\) is the causality relation between events which is partial order
- \(\# \subseteq E \times E\) is a binary conflict relation between events which is irreflexive and symmetric
- \(I : E \to Act\) is the labeling function mapping events to actions

1. Causality relation satisfies principle of finite causes:
   \[\forall e, e' \in E : \{e' \mid e' \leq e\}\text{ is finite.}\]

2. Conflict relation satisfies principle of conflict heredity:
   \[\forall e, e', e'' \in E : e\#e' \land e \leq e'' \Rightarrow e'\#e''\]

3. A computation \(x\) is a sequence of events such that
   - \(e' \in x \land e \leq e' \implies e \in x\)
     (if an event has occurred then all events on which it causally depends have occurred too)
   - \(\forall e, e' \in x. \neg(e\#e')\)
     (no two conflicting events can occur together in the same computation)

Overview of Dynamic Condition Response Structures

Limitation of events structures
- finite representation of infinite behavior
- accepting condition
- distribution of events

Causality relation
- condition and response relations

Event Structures

Dynamic Condition Response Structures

Distributed Dynamic Condition Response Structures

Binary conflict relation
- dynamic include and exclude relations
  + repeated execution of events
  + distribution of events
Dynamic Condition Response Structures (DCRS)

A *dynamic condition response structure* (DCR) is a tuple \( D = (E, \text{Act}, \rightarrow \bullet, \bullet \rightarrow, \pm, l) \) where

1. Causality relation \((\leq)\) is replaced with two relations: condition \((\rightarrow \bullet)\) and response \((\bullet \rightarrow)\) relations.
2. If \( e \rightarrow \bullet e' \), then \( e \) must happen before \( e' \) as a pre-condition.
3. If \( e \bullet \rightarrow e' \), then after \( e \) happens, \( e' \) must eventually happen or be excluded indefinitely for the computation to be accepting.
4. Conflict relation \((\#)\) is generalized to include \((\rightarrow +)\) and exclude \((\rightarrow \%)\) relations to include/exclude events dynamically.
5. \( \pm(e, e') = + \) if event \( e \) gets executed, then it will include event \( e' \)
6. \( \pm(e, e') = \% \) if event \( e \) gets executed, then it will exclude event \( e' \)
7. Conflict relation is monotone, but dynamic inclusion/exclusion allows an event to alternate between being in conflict and not.
8. Execution of an event only depends on the condition relation (restricted to the currently included events), where acceptance of a computation depends on response relation.
Distributed Dynamic Condition Response Structures*

**Definition**

A *distributed* dynamic condition response structure (DDCR) is a tuple

\[(E, \text{Act}, \rightarrow\bullet, \bullet\rightarrow, \pm, I, R, P, \text{as})\]

where \((E, \text{Act}, \rightarrow\bullet, \bullet\rightarrow, \pm, I)\) is a dynamic condition response structure, \(R\) is a set of *roles*, \(P\) is a set of *principals* (e.g. persons/processors/agents) and \(\text{as} \subseteq (P \cup \text{Act}) \times R\) is the role assignment relation to executors and actions.

- assigning roles to actions provide granularity of permissions
- assigning principals to roles gives the permission to execute actions

DCRS Execution semantics for Finite Runs

Definition
For a distributed DCR \( D = (E, \text{Act}, \rightarrow, \cdot \rightarrow, \pm, I, R, P, \text{as}) \) the corresponding labelled transition systems \( T(D) \) to be the tuple \((S, (\emptyset, E, \emptyset), \rightarrow \subseteq S \times \text{Act} \times S)\) where \( S = \mathcal{P}(E) \times \mathcal{P}(E) \times \mathcal{P}(E) \) is the set of states, \((\emptyset, E, \emptyset) \in S\) is the initial state, \(\rightarrow \subseteq S \times (P \times \text{Act} \times R) \times S\) is the transition relation given by

\[
(E, I, R) \xrightarrow{(e, p, a, r)} (E \cup \{e\}, I', R') \qquad\text{where}
\]

- \( e \in I, l(e) = a, p\text{ as }r,\text{ and }a\text{ as }r \)
- \( \{e' \in I \mid e' \rightarrow \cdot e\} \subseteq E \)
- \( I' = (I \cup \{e' \mid \pm(e, e') = +\}) \setminus \{e' \mid \pm(e, e') = -\} \)
- \( R' = (R \setminus \{e\}) \cup \{e' \mid e \rightarrow \cdot e'\} \)

- map to a labelled transition system to defined accepting runs
- states of transition system will be \((E, I, R)\) where \(E \subseteq E\) is set of happened events, \(I \subseteq E\) represents set of currently included events, \(R \subseteq E\) represents set of pending response events
- first condition says that only currently include events can be executed, the label \((p, a, r)\) says that the label of event \(e\) must be \(a\), which must be assigned to a role \(r\) and principal \(p\)
- second condition says that all condition events to \(e\) must have been executed
- third and fourth conditions are updates to sets of included and pending responses.
- **Accepting State:** Any state with no pending responses \((R \cap I = \emptyset)\)
- **Accepting run:** Any run which is ending with accepting state \((R \cap I = \emptyset)\)
DCRS Graphical Notation- Health Care Workflow

Rules:

- doctor has to sign whenever he prescribes a medicine
- medicine can be given only after doctor’s sign
- nurse can check medicine and say that “I don’t trust medicine”, in that case doctor has to either to sign again or change medicine and sign.

\[
E = \{pm, s, gm, dt\}
\]

\[
\rightarrow = \{(pm, s), (s, gm), (s, dt)\}
\]

\[
\rightarrow\bullet = \{(pm, s), (pm, gm), (dt, ss)\}
\]

\[
\rightarrow+ = \{(s, gm), (s, dt)\}
\]

\[
\rightarrow\% = \{(gm, dt), (dt, gm)\}
\]

- **OK and Accepting**
  - ✓ [pm, s, gm, gm]
  - ✓ [pm, s, dt, s, gm]
  - ✓ [pm, s, dt, pm, s, gm]
  - ✓ [pm, pm, s, gm]
  - ✓ [pm, s, pm, s, gm]

- **Not Possible**
  - ○ [pm, gm]
  - ○ [pm, s, dt, gm]

- **Possible, but not Accepting**
  - ○ [pm, s, gm, pm] (pending : gm, s)
  - ○ [pm, s, dt] (pending : s)
• In the first example a run $a^\omega$ should be accepting.
• In the second example, a run $a^* b^* (ab)^\omega$ should be accepting.
• But strong accepting condition ($R \cap I = \emptyset$) does not allow those runs accepting.
DCRS Execution semantics for Infinite Runs*

For a finite distributed DCR $D = (E, Act, \rightarrow, \cdot, \cdot, l, R, P, as)$ where $E = \{e_1, \ldots, e_n\}$ we define the corresponding Büchi-automaton $Aut(D)$ to be the tuple $(S, s, \rightarrow \subseteq S \times Act \times S, F)$ where $S = P(E) \times P(E) \times P(E) \times \{1, \ldots, n\} \times \{0, 1\}$ is the set of states and $s = (\emptyset, E, \emptyset, 1, 1) \in S$ is the initial state and $F = P(E) \times P(E) \times P(E) \times \{1, \ldots, n\} \times \{1\}$ is the set of final or accepting states. $\rightarrow \subseteq S \times (P \times Act \times R) \times S$ is the transition relation given by

$$(E, l, R, i, j) \xrightarrow{(p,a,r)} (E \cup \{e\}, l', R', i', j')$$

- Semantics of $E, l, R$ are same as previous accepting condition
- $j' = 1$ if
  1. $l' \cap R' = \emptyset$
  2. $min(M) \in (l \cap R' \setminus (l' \cap R')) \cup \{e\}$
  3. $M = \emptyset$ and $min(l \cap R) \in (l \cap R' \setminus (l' \cap R')) \cup \{e\}$
  otherwise $j' = 0$.
- $i' = rank(min(M))$ if $min(M) \in (l \cap R' \setminus (l' \cap R')) \cup \{e\}$
- $i' = rank(l \cap R)$ if $M = \emptyset$ and $min(l \cap R) \in (l \cap R' \setminus (l' \cap R')) \cup \{e\}$
- $i' = i$ otherwise.

for $M = \{e \in l \cap R \mid rank(e) > i\}$.

- Mapped to a Generalized Büchi automata
- Instead of looking for $R \cap l = \emptyset$, we make sure that all pending response events $(R \cap l)$ will be eventually executed infinitely often.
- In other words, infinite run is not accepting if one or more pending response event(s) will never executed.

Accepting condition for infinite runs - example

Infinite runs: Büchi Automaton for example

Accepting run for case where both a and b are executed indefinitely often with i copies of state
Model checking and Verification of DCRS processes

✦ Declarative models offer little support and hard to perceive.
✦ Model checking/verification enhances reliability and trustworthiness among processes.

Model checking with Spin model checker:
✓ A prototype application with interface to Spin model checker
✓ DCRS model can be automatically translated to Promela code.
✓ Both safety and liveliness properties (expressed in LTL) can be verified for both finite and infinite runs.
✓ State space in Spin for DCRS models is un-necessarily large due to limited language constructs in Promela

Model checking with Zing model checker:
✓ Prototype application upgraded to generate code for Zing.
✓ State space in Zing is much smaller than Spin due to rich language constructs supported by Zing
✓ Only safety properties can be verified.
Runtime Monitoring of DCRS processes (work at MSRI)

✦ Verification of properties can be expensive for large process specifications.
✦ Runtime monitoring of DCRS processes makes sure that certain desired properties are preserved during execution.
✦ A predefined set of property specification patterns can be used to specify properties.
✦ Properties transformed to a finite state machine.

DCRS Prototype Implementation

Roadmap for next 18 months

1) What are the formal semantical models suitable for describing flexible workflow processes for health care and other dynamic services?

✓ Extensions to formal model
  ✓ Data, Time,
  ✓ sub-processes,
  ✓ Exceptions and Compensation
✓ Support for quantitative workflows(??)

3) What are the suitable model checking and verification techniques for enhancing trustworthiness of declarative, quantitative and dynamic workflows?

✓ DCRS prototype implementation to updated with extensions to formal model
✓ Support for more model checking and verification tools

Last 3 slides! Please be patient...
Roadmap for next 18 months

2) How should one describe interfaces, contracts and interactions for declarative, quantitative and dynamic workflows?

✓ A choreography/contract describes a global view on how different participants interact.
✓ Session Types studied how to project such global interactions to individual participants called End-point projections\(^1\).
✓ An interface to workflow is the observable behavior from outside (similar to choreography/contract)
✓ Extensions to theory of end-point projections will be studied to include quantitative and declarative contracts.

Expected Contribution

1. Contribution to Theory
   ✓ A formal process model for flexible declarative workflows based on inspiration from
     ✷ Resultmaker workflow method
     ✷ Declare
   ✓ Suitable extensions to theory of end-point projections for declarative and quantitative contracts.

2. Contribution to Practice
   ✓ New ideas/methods from thesis will provide the foundations for development of real time workflow systems for complex/rapidly changing sectors, where process flexibility is important.
   ✓ Prototype implementation based on formal model will serve as good model/proof of concept, for application of model checking and verification techniques to real-time workflows.


Thank You
Questions & Comments ?