Composition and Semantic Enhancement of Web-Services
Introduction to the CASheW-s Project

• Our main objective is to develop a more generic approach to Web-Service composition.

• Therefore we are investigating the use of a timed process calculus to provide *compositional* behavioural semantics for workflows.

• The culmination of this will be a workflow engine, which will first be able to orchestrate OWL-S workflows.

• In this presentation we look at the operational semantics for OWL-S, and our approach to building them.
CaSHew-NUtS

• A conservative extension of the timed process calculus CaSE, which itself is a conservative extension of Milner's CCS.

• Extends CCS with the notion of abstract clocks, which facilitate multi-party synchronization.

• In CaSE, clocks are bound by maximal progress, meaning silent actions always take precedence over clock ticks.

• CaSHew-NUtS extends this concept with the possibility of clocks which do not exhibit maximal progress.
CASeHW-s Architecture

Language extensions

CASHeW-s Editor

Typecheck Workflow

CASHeW-s Engine

Evaluate Functions

Haskell Evaluator Service

Import types

OWL-S Publishing Gateway

Published Workflow

XSD Type DB

Compiled Workflow Processes

\( p_1 \), \( p_2 \), \( p_3 \)

SOAP Endpoints
CaSHew-NUtS Composition Rules

Com3 \[
\frac{E \xrightarrow{\alpha} E', F \xrightarrow{\bar{\alpha}} F'}{E | F \xrightarrow{\tau} E' | F'} \Lambda = \{a, b, c, \ldots\} \quad \bar{\Lambda} = \{\bar{a}, \bar{b}, \bar{c}, \ldots\}
\]
\[\alpha \in \mathcal{A} = \Lambda \cup \bar{\Lambda} \cup \{\tau\}\]

Com1 \[
\frac{E \xrightarrow{\alpha} E'}{E | F \xrightarrow{\alpha} E' | F}
\]

Com2 \[
\frac{F \xrightarrow{\alpha} F'}{E | F \xrightarrow{\alpha} E | F'}
\]

Com4 \[
\frac{E \xrightarrow{\sigma_i} E' \quad F \xrightarrow{\sigma_j} F'}{E | F \xrightarrow{\sigma_i \cdot \sigma_j \cdot k} E' | F'} \quad \gamma \in \mathcal{A} \cup \mathcal{T}
\]
\[\mathcal{T} = \{\rho, \sigma, \ldots\}\]
CASheW-s Syntax

• Problems with OWL-S Syntax
  – Incoming dataflow tied to Performance restricting further composition.
  – Fine for persistence/communication, but doesn't represent the composition of a system.
  – Uncomfortable notion of Produce tied to dummy variable TheParentPerform.

• CASheW-s syntax
  – More open to composition.
  – Allows compositional translation from OWL-S syntax.
Process Syntax for CASHeW-s

\[
Process ::= \text{AtomicProcess} \ m \ \text{AProcess} | \text{CompositeProcess} \ m \ \text{CProcess} \\
\quad \text{ConsumeList ProduceList} \\
CProcess ::= \text{Sequence} \ \text{PerformanceList} | \text{Split} \ \text{PerformanceList} | \text{SplitJoin} \ \text{PerformanceList} | \text{Any-Order} \ \text{PerformanceList} | \text{ChooseOne} \ \text{PerformanceList} | \text{IfThenElse} \ \text{Performance} \ \text{Performance} | \text{RepeatWhile} \ \text{Performance} | \text{RepeatUntil} \ \text{Performance}
\]
Performance Syntax for CASHeW-s

\[ \text{Performance} ::= \text{Perform } n \text{ Process DataAggregation} \]
\[ \text{Connection} ::= \text{Connect } n \text{ o a j} \]
\[ \text{PerformanceList} ::= \text{Performance} | \]
\[ (\text{PerformanceList}) ; \text{Performance} | \]
\[ (\text{PerformanceList}) ; \text{Connection} \]
\[ \text{DataAggregation} ::= \text{ValueDataList} \]
\[ \text{ValueCollectorList} \]
\[ \text{ValueData} ::= \text{ValueData a} \]
\[ \text{ValueDataList} ::= \epsilon | \text{ValueData ValueDataTail} \]
\[ \text{ValueDataTail} ::= \epsilon | ; \text{ValueData ValueDataTail} \]
\[ \text{ValueCollector} ::= \text{ValueCollector a k} \]
\[ \text{ValueCollectorList} ::= \epsilon | \text{ValueCollector ValueCollectorTail} \]
\[ \text{ValueCollectorTail} ::= \epsilon | ; \text{ValueCollector ValueCollectorTail} \]
\[ \text{Consume} ::= \text{Consume a n b j} \]
\[ \text{ConsumeList} ::= \epsilon | \text{Consume ConsumeTail} \]
\[ \text{ConsumeTail} ::= \epsilon | ; \text{Consume ConsumeTail} \]
\[ \text{Produce} ::= \text{Produce c n d} \]
\[ \text{ProduceList} ::= \epsilon | \text{Produce ProduceProduceTail} \]
\[ \text{ProduceTail} ::= \epsilon | ; \text{Produce ProduceProduceTail} \]
Orchestration Channels

- **r** is the *ready to execute* channel, which a process uses to indicate that it has no further execution pre-conditions. (Something the informal semantics rely on, but no-one else has formalised).

- **e** is the *permission to execute* channel, which a process must receive input on before it can begin executing.

- **t** signifies the *token*, which signifies permission to execute for each of the process's child performances (in a similar fashion to a token ring network). Different token passing games facilitate performance serialization.
Orchestration Clocks

- Two main clock types used for orchestration.
- $\sigma^m$ is the *process clock*, it ranges over the entire scope of the process and its child performances and may be used to resynchronize (such as after a split-join), where $m$ is the name of the process.

- $\sigma^n, \sigma^o$ are *performance clocks*, they are used to signal that a performance has completed, and are used to decide when control can be passed on to another scheduler in the system, where $n$ and $o$ are names of performances.
Composite Process Layout

CompositeProcess $c_p$

Sequence
- Perform $n_1 p_1$
- Connect $c n_1 a n_2 0$
- Connect $d n_1 a n_3 2$
- Perform $n_2 p_2$
  - ValueData $a$
- Connect $c n_2 a n_3 1$
- Perform $n_3 p_2$
  - ValueCollector $a 2$
- Consume $a n_1 a 0$
- Produce $c n_3 c$
OWL-S Process Semantics

\[
\text{[AtomicProcess } m \text{ P}]_C^A = m [P]_C^A
\]

\[
\text{[CompositeProcess } m \text{ P G H}]_C^A = (m [P]_C^{Am} \mid [G]_C^A \mid [H]_C^\emptyset) \setminus A^m \cup C^m / \{\sigma^c \mid c \in C\}
\]

Where
- m is a process name
- p is a process
- A is a set of inputs
- C is a set of outputs
- G is a \textit{Consume List}
- H is a \textit{Produce List}
Example Atomic Process Semantics

\[
\llbracket An\text{AtomicProcess} \rrbracket \{a_1, a_2\} =
\mu X. < a_1, a_2 > . \overline{r}. e. \tau. \overline{c}. X
\]
Consume Semantics

- *Consume* pulls an input which is required to run a process.

\[
[\text{Consume } a \ n \ b \ j]_{\{a\}} = \mu X. a. b^n_j . X
\]

- Wires like Consume, patiently wait for input and then insistently output.
Produce Semantics

- *Produce* pushes an output which has been produced by a process.

\[
\begin{align*}
\left[\text{Produce } c \ n \ d\right]\{c\} &= \mu X. d^n \cdot \overline{c} \cdot X
\end{align*}
\]

- Within CASHeW-s, *Produce* is not a type of performance, rather a type of connection.
Connection Semantics

• *Connect* shunts the output of one performance in a composite process, to the input of another.

\[
[\text{Connect } n \ c \ o \ a \ j] = \mu X. c^n \overline{a_j^o}.X
\]
Composite Process Semantics

- Defined in terms of a top-level Governor process, and in the case of unbounded child-performances an inductively defined context-based composition semantics, which pair a Scheduler with the performance semantics.

\[ m[\text{Sequence } Q]^A_C = m[\text{seq } Q]^A_C / \sigma^m \setminus t \]

\[ m[\text{SplitJoin } Q]^A_C = (m[\text{sj } Q]^A_C | \mu \text{X.} \sigma^m \cdot \overline{e}.\sigma^m.\sigma^m.\text{X})//\sigma^m \]

\[ m[\text{AnyOrder } Q]^A_C = m[\text{any } Q]^A_C / \sigma^m \setminus t \]
Sequence Semantics
Base Case

\[ m \left[ \text{seq Perform } n \ p \ U \ V \right] _C^A = \]
\[ (n [\text{Perform } n \ p \ U \ V] _C^n [e \leftrightarrow e^i, r \leftrightarrow r^i] | \]
\[ \mu X. r^i . \overline{r}. e.e^i. \sigma^n \sigma_m [\overline{t}. \sigma^m.X] \sigma^m(X))/\sigma^n \setminus \{r^i, e^i\} \]
Sequential Composition Semantics
General Case

\[ m[^{seq}(Q); \text{Perform } n \ p \ U \ V]^A_{CQ \cup Cn} = \]
\[(n[^{Perform} n \ p \ U \ V]^A_{Cn}[e \leftrightarrow e^i, r \leftrightarrow r^i] \mid m[^{seq}Q]_{CQ}^A[t \leftrightarrow t^i] \mid \]
\[\mu X.t^i.r^i.e^i.\sigma^n_{\sigma_m}[[\overline{t}.\sigma^m.X] \sigma^m(X))]/\sigma^n \setminus \{r^i, e^i\}\]
AnyOrder Composition Semantics
Base Case

\[ m \left[ \text{any} \right. \text{Perform } n \ p \ U \ V \left] \right. C^A = (m \left[ \text{any} \right. \text{Perform } n \ p \ U \ V \left] \right. C^A[e \mapsto e^i, r \mapsto r^i] | \]

\[
\mu X.r_i^m \cdot (r.e.e^i.\sigma^n_{\sigma^m}.[t.\sigma^m.X] \sigma^m(X) \quad + \quad t.e^i.\sigma^n_{\sigma^m}.[t.\sigma^m.X] \sigma^m(X))) \setminus \{e^i, r^i\} / \sigma^n
\]
AnyOrder Composition Semantics
General Case

\[ m \left[ \text{any} (Q); \text{Perform} \ n \ p \ U \ V \right]^A_{Q \cup A^n} = \]
\[ m \left[ \text{any} \ \text{Perform} \ n \ p \ U \ V \right]^A_{C^n} \mid m \left[ \text{any} Q \right]^A_{C Q} \]

- We use this induction in all cases to define the semantics for the general case where all performances are handled in the same way.
Split/SplitJoin Process Semantics (Governor)

\[
m\left[\text{SplitJoin } Q\right]^A_C = (m\left[\text{sj } Q\right]^A_C | \mu X.\sigma^m.\overline{r}.e.\sigma^m.\sigma^m.X) \parallel \sigma^m
\]

\[
m\left[\text{Split } Q\right]^A_C = (m\left[\text{split } Q\right]^A_C | \mu X.\sigma^m.\overline{r}.e.\sigma^m.\sigma^m.X) \parallel \sigma^m
\]
SplitJoin Composition Semantics

\[ m[\text{splitJoin}\ n\ p\ U\ V]^{A}_{C} = (m[\text{splitJoin}\ n\ p\ U\ V]^{A}_{C}[e \leftrightarrow e^{i},\ r \leftrightarrow r^{i}] | \mu X. r^{i}_{\sigma^m}. \sigma^m. \sigma^m. \overline{e^{i}}. \sigma^{n}_{\sigma^m}. \sigma^m. X) \setminus \{e^{i}, r^{i}\} / \sigma^{n} \]
Split Composition Semantics

\[ m \left[ \text{split Perform } n \ p \ U \ V \right]^A_C = \left( m \left[ \text{split Perform } n \ p \ U \ V \right]^A_C[e \leftrightarrow e^i, r \leftrightarrow r^i] \right) | \mu X. r^i_{\sigma_m}. \sigma^m. X ) \ \setminus \{e^i, r^i\} / \sigma^n \]

- Split is our primary motivation for clock ticks not bound by maximal progress
Next Step: Haskell Implementation

- We already have an implementation of the CaSHew-NUtS Process Calculus in Haskell, the next step is to define semantics for mapping OWL-S to this representation.

- The Haskell implementation allows the calculus to be grounded in IO operations, enabling Web-Service invocation.

- This can then be combined with our HAIFA interoperability kit to enable orchestration.
Conclusion

- We have presented a timed process calculus semantics for OWL-S, which we will shortly be using to build an orchestration engine.
- We predict that this approach to providing operational semantics can be applied to other work-flow languages, allowing a single engine to be able handle heterogeneous orchestration.
- All of this will be combined with the safety of Haskell, to build reliable, predictable workflows.
More to come soon...
Basic CCS Rules

**Act**
\[ \alpha.E \xrightarrow{\alpha} E \]

**Sum1**
\[ E \xrightarrow{\alpha} E' \quad \frac{E + F \xrightarrow{\alpha} E'}{E + F} \]

**Sum2**
\[ F \xrightarrow{\alpha} F' \quad \frac{E + F \xrightarrow{\alpha} F'}{E + F} \]

**Com1**
\[ E \xrightarrow{\alpha} E' \quad \frac{E | F \xrightarrow{\alpha} E' | F}{E | F} \]

**Com2**
\[ F \xrightarrow{\alpha} F' \quad \frac{E | F \xrightarrow{\alpha} E | F'}{E | F} \]

**Com3**
\[ E \xrightarrow{\alpha} E', F \xrightarrow{\bar{\alpha}} F' \quad \frac{E | F \xrightarrow{\tau} E' | F'}{E | F} \]

**Res**
\[ E \xrightarrow{\gamma} E' \quad \frac{E \setminus a \xrightarrow{\gamma} E' \setminus a}{E \setminus a} \quad \gamma \notin \{\alpha, \bar{\alpha}\} \]

**Rec**
\[ \mu X. E \xrightarrow{\gamma} E' \quad \frac{\mu X. E \xrightarrow{\gamma} E' \{\mu X. E / X\}}{\mu X. E} \]
CaSE Additions to CCS

Idle: $0 \xrightarrow{\sigma_1} 0$

Patient: $a.E \xrightarrow{\sigma_1} a.E$

Stall: $\Delta_\sigma \xrightarrow{\rho_1} \Delta_\sigma$

TO1: $[E]_\sigma(F) \xrightarrow{\sigma_i} F$

TO2: $E \xrightarrow{\gamma} E'$

TO3: $[E]_\sigma(F) \xrightarrow{\gamma} E'$

where:
1) $\rho \neq \sigma$
2) $\nabla i \cdot \gamma = \sigma_i$

and:
a) $i = 0$ if $\tau \in IA(E)$, 1 otherwise
b) $k = 0$ if $\tau \in IA(E | F)$, 1 otherwise
c) $\sigma_1 \notin IA(E)$
d) $\nabla i \cdot \sigma_i \in IA(E)$
CaSE Additions (cont)

\[
\begin{align*}
\text{STO1} & \quad [E] \sigma(F) \xrightarrow{\sigma_i} F \\
\text{STO3} & \quad E \xrightarrow{\sigma_i} E' \\
\text{STO2a} & \quad E \xrightarrow{\alpha} E' \\
\text{STO2b} & \quad E \xrightarrow{\rho_i} E' \\
\text{Com4} & \quad E \xrightarrow{\sigma_i} E' \quad F \xrightarrow{\sigma_j} F' \quad E \| F \xrightarrow{\sigma_i \cdot j \cdot k} E' \| F'
\end{align*}
\]

where:
1) $\rho \neq \sigma$
2) $\exists i \cdot \gamma = \sigma_i$

and:

- a) $i = 0$ if $\tau \in \mathcal{IA}(E)$, 1 otherwise
- b) $k = 0$ if $\tau \in \mathcal{IA}(E \| F)$, 1 otherwise
- c) $\sigma_1 \notin \mathcal{IA}(E)$
- d) $\exists i \cdot \sigma_i \in \mathcal{IA}(E)$
CaSE Additions (cont)

Hid1\[\frac{P \xrightarrow{\sigma_1} P'}{P/\sigma \xrightarrow{\tau} P'/\sigma}\]

Hid2\[\frac{P \xrightarrow{\alpha} P'}{P/\sigma \xrightarrow{\alpha} P'/\sigma}\]

Hid3\[\frac{P \xrightarrow{\rho_i} P'}{P/\sigma \xrightarrow{\rho_i} P'/\sigma}\]

1, c

where:
1) $\rho \neq \sigma$
2) $i \cdot \gamma = \sigma_i$

and:

a) $i = 0$ if $\tau \in \mathcal{IA}(E)$, 1 otherwise
b) $k = 0$ if $\tau \in \mathcal{IA}(E \mid F)$, 1 otherwise
c) $\sigma_1 \notin \mathcal{IA}(E)$
d) $\exists i \cdot \sigma_i \in \mathcal{IA}(E)$
CaSHew-NUtS

UHid1
\[
\frac{P \xrightarrow{\sigma_i} P'}{P \parallel \sigma \xrightarrow{\tau} P' \parallel \sigma}
\]

UHid2
\[
\frac{P \xrightarrow{\alpha} P'}{P \parallel \sigma \xrightarrow{\alpha} P' \parallel \sigma}
\]

UHid3
\[
\frac{P \xrightarrow{\rho_i} P'}{P \parallel \sigma \xrightarrow{\rho_i} P' \parallel \sigma}
\]

where:

1) \( \rho \neq \sigma \)

and:

a) \( i = 0 \) if \( \tau \in \mathcal{I}(E) \), 1 otherwise

b) \( k = 0 \) if \( \tau \in \mathcal{I}(E \mid F) \), 1 otherwise

c) \( \sigma_1 \notin \mathcal{I}(E) \)

d) \( i \cdot \sigma_i \in \mathcal{I}(E) \)