A Compositional Operational Semantics for OWL-S

Barry Norton[1,2], Simon Foster[2], and Andrew Hughes[2]

[1] Knowledge Media Institute, Open University, UK.
[2] Department of Computer Science, University of Sheffield, UK.
e-mail: b.norton@dcs.shef.ac.uk

Abstract. Software composition via workflow specifications has received a great deal of attention recently. One reason is the high degree of fit with the encapsulation of software modules in service-oriented fashion. In the Industry, existing workflow languages have been merged to form WS-BPEL, the Business Process Execution Language for Web Services. In the Research community OWL-S, a ontology for web services, has been submitted for standardisation alongside OWL, the Web Ontology Language in which it is expressed. The OWL-S Process Model is based on an abstraction of the common features of industrial workflow languages. On the one hand, WS-BPEL has only informal semantics; on the other, the type of semantics given to ontology-based work tends to be structural rather than computationally oriented. As a result the semantics developed for DAML-S, which led to OWL-S, are still deficient in some regards. In this paper we shall survey the existing semantics and introduce a novel semantics for the latest version of OWL-S that is focussed on the principle of compositionality, so far not tackled.

1 Introduction

A recent article in the inaugural editorial of the *International Journal on Semantic Web and Information Systems* [12] reviewed the properties necessary for work on semantics for the Semantic Web. One fundamental proposed explained was that of *compositionality*. While well understood by the formal methods community as a property that should apply to *behavioural* semantics, the time is right to make this point, and present means to achieve it, to the semantic web (services) community. Generally stated, this principal means that where semantics are given to a formal language, any respective members of the equivalence classes of two terms should semantically compose into the same equivalence class as the semantics given to the composite expressed in the target language. There are two important *practical* consequences that we should like to draw out in the context of behavioural semantics for service composition. The first is the ability to form a semantic model for an orchestration, step by step matching the workflow by which it is syntactically expressed, as this is built via interaction with an editor. The point is to avoid rebuilding the semantic model from scratch at each step. By rather extending the model *incrementally*, we can hope to represent semantic properties to the user in real time.

The second consequence relates to the principal of *substitutability*, to which compositionality leads. Practically this allows us to take any member of the equivalence class of the semantics for a term to represent that term as further composition takes place. This allows us to somewhat avoid the state explosion problem, fundamental to concurrent systems, by abstracting from internal states at each point encapsulation takes place, having shown that a so-called *observational equivalence* that we shall review later, is a congruence in our intermediate syntax for semantic translation. Such an observational theory, allowing this abstraction, is the basis of the process calculus CCS [5] which is extended with mobility to form the Pi-Calculus [6], the inspiration for much work in workflow-oriented service composition.

Our previous work [8] has demonstrated the compositional modelling of dataflow-oriented software composition using a novel process calculus CaSE, a conservative extension of CCS. This calculus develops on the tradition of encoding the progress of time qualitatively via abstract ticks of a clock related to communication behaviour via the principle of maximal progress [4], *i.e.* where silent actions are preemptive over clock transitions. CaSE introduced the concept of setting scopes under which behaviour is measured by a specific clock, running at a different rate from other such, via an operator that at once 'hides' that clock, *i.e.* makes it both immune from the preemptive effects of outsiders' silent actions and unobservable to outsiders, and makes it preemptive over the clocks that are still open in the outside.

Our previous model was for systems where scheduling is governed in a data-driven fashion according to generalised dataflow graphs and in a serialised fashion, *i.e.* with course-grained interleavings so that the execution of each actor is atomic. In OWL-S this execution model is just one (specifically the *AnyOrder* process type) of several in an algebraic definition of workflow-oriented *processes* — analogous to 'components' in our previous work — defined hierarchically in terms of *performances* — analogous to our previous 'component instances' [3].

Rather than defining a compositional model for a grammar directly based on the Process Model part of the OWL-S ontology, we define a derived formal language, which we called CASheW-S (named for our project on the *Composition And Semantic Enhancement of Web Services*) that has a greater degree of 'composability' than this. In particular we seperate a first class notion of *connection* from the definition of performance. In OWL-S, performances have to be declared with their complete inbound dataflow attached. In CASheW-S, performances and connections between them can be seperately composed in the same way as they are in a graphical editor.

In the following section we will complete and explain our adapted syntax and review its informal semantics. In Section 3 we will present our syntax and operational semantics for a conservative extension to CaSE, which we name *CaSHew-NUtS* for *Calculus for Synchronous Hierarchies extended with Non-deterministic and Un-timed Synchronisations*. With this in place we can present a translation from our CASheW-S syntax into CaSHew-NUtS in Section 4. We compare this to existing approaches in Section 5 and conlude in Section 6.

2 CASheW-S Syntax

As shown in Table 1, processes in CASheW-S are either atomic or composite. Both are named from a set we chose, for the purposes of the semantics, to range over with m. Composite processes are defined in terms of performances of other processes each of which is given a name that, for our purposes, we allow to be ranged over by n and o and, like process names, must be guaranteed unique.

$$
\begin{aligned}
Process &::= \textbf{AtomicProcess}\ m\ AProcess \mid \\
&\quad\ \textbf{CompositeProcess}\ m\ CProcess \\
&\qquad ConsumeList\ ProduceList \\
CProcess &::= \textbf{Any-Order}\ PerformanceList \mid \\
&\quad\ \textbf{Sequence}\ PerformanceList \mid \\
&\quad\ \textbf{Split}\ PerformanceList \mid \\
&\quad\ \textbf{SplitJoin}\ PerformanceList \mid \\
&\quad\ \textbf{ChooseOne}\ PerformanceList \mid \\
&\quad\ \textbf{IfThenElse}\ Performance\ Performance \mid \\
&\quad\ \textbf{RepeatWhile}\ Performance \mid \\
&\quad\ \textbf{RepeatUntil}\ Performance \\
Performance &::= \textbf{Perform}\ n\ Process\ DataAggregation \\
Connection &::= \textbf{Connect}\ n\ c\ o\ a\ j \\
PerformanceList &::= Performance \mid \\
&\quad\ (PerformanceList); Performance \mid \\
&\quad\ (PerformanceList); Connection \\
DataAggregation &::= ValueDataList \\
&\quad\ ValueCollectorList \\
ValueData &::= \textbf{ValueData}\ a \\
ValueDataList &::= \epsilon \mid ValueData\ ValueDataTail \\
ValueDataTail &::= \epsilon \mid ;\ ValueData\ ValueDataTail \\
ValueCollector &::= \textbf{ValueCollector}\ a\ k \\
ValueCollectorList &::= \epsilon \mid ValueCollector\ ValueCollectorTail \\
ValueCollectorTail &::= \epsilon \mid ;\ ValueCollector\ ValueCollectorTail \\
Consume &::= \textbf{Consume}\ a\ n\ b\ j \\
ConsumeList &::= \epsilon \mid Consume\ ConsumeTail \\
ConsumeTail &::= \epsilon \mid ;\ Consume\ ConsumeTail \\
Produce &::= \textbf{Produce}\ c\ n\ d \\
ProduceList &::= \epsilon \mid Produce\ ProduceTail \\
ProduceTail &::= \epsilon \mid ;\ Produce\ ProduceTail
\end{aligned}
$$

Table 1. The CASheW-S Process Type

When it comes to the declaration of dataflow, there are two differences from OWL-S, but each allows a direct and compositional translation from the constructs there. We consider first the *Connection* syntax, introduced with the keyword **Connect**. This is a *first class* equivalent to the more restrictive **ValueSource** construct, as well as providing part of the role of the **ValueFunction** construct, in OWL-S. This also allows us to clarify the role of the **Produce** construct, which we cast as a specialisation to these connections rather than a specialisation of performance as in OWL-S, and to introduce the dual **Consume** construct.

ValueSource and ValueFunction declarations in OWL-S are strictly tied to performances, in particular their implicit destinations, meaning that a performance must declare explicitly its complete in-coming dataflow when composed into a system, and can not be the subject (in the role of destination) to any further dataflow. In CASheW-S we should like to represent the degree of composition appropriate to an interactive editor and so allow connections to be composed as first class entities between any existing performances. As such they must identify two performances, n and o, and respectively the output c of the destination, and the input a of the source. It also declares a (numbered) component of the input, which is to be supplied, j.

Performances can split the inputs of the process being performed into components via the *ValueCollector* construct. This allows the second function of OWL-S *ValueFunction*s to be represented (the actual definition of the associated function is elided, just as it is as an XML literal in OWL-S, but we must know how many communications are needed). Whereas OWL-S performances contain *ValueData*, *ValueFunction* and *ValueSource* declarations, their associated *DataAggregation* construct in CASheW-S contain only *ValueData* and *ValueCollector* declarations. The dataflow that provides the input components of the value collectors, as well as the other inputs not provided as constants via value data declarations (implicitly having only a singleton component, numbered 0), is defined via connections. In our semantics these will become atomic names for channels c^n and a^n_j respectively. OWL-S performances can be compositionally translated since the implicit connections can be immediately composed with the CASheW-S performance.

In order to define a composite process two different type of connections are needed. To define a prototypical input, a, this must be associated with a component, b^n_j, of a performance input (associated with the prototypical input, b, defined by the process performed) of some component performance, n. This type of connection is introduced with the keyword **Consume**. The keyword **Produce**, unlike the one in OWL-S, is the direct dual to this, connecting a performance output, d^n, to the prototypical output, c, of the enclosing composite process. Neither of these constructs requires the use of the poorly named so-called dummy variable 'theParentPerform' used in OWL-S, but can be translated directly from such OWL-S *Produce* and *ValueSource* declarations.

3 CaSHew-NUtS

To provide an operational semantics for the CASheW-S language, we translate each term into the process calculus CaSHew-NUtS, for which the core syntax is defined in Table 2.

$$\mathcal{E} ::= \mathbf{0} \mid \Delta \mid \Delta_\sigma \mid \alpha.\mathcal{E} \mid \llbracket \mathcal{E} \rrbracket \sigma(\mathcal{E}) \mid \llparenthesis \mathcal{E} \rrparenthesis \sigma(\mathcal{E}) \mid \mathcal{E} + \mathcal{E} \mid \mathcal{E} | \mathcal{E} \mid \mathcal{E}[a \mapsto b] \mid \mathcal{E} \setminus a \mid \mathcal{E}/\sigma \mid \mathcal{E} /\!\!/ \sigma \mid \mu X.\mathcal{E} \mid X$$

Table 2. Core CaSHew-NUtS Syntax

$$
\begin{array}{rcl}
a, \overline{a}, b, \overline{b}, \cdots \in \Lambda \cup \overline{\Lambda} & & \rho, \sigma, \cdots \in \mathcal{T} \\
L \subseteq \Lambda & & T \subseteq \mathcal{T} \\
\mathcal{A} = \Lambda \cup \overline{\Lambda} \cup \{\tau\} \qquad \mathcal{L} = \mathcal{A} \cup \mathcal{C} & & \mathcal{C} = \mathcal{T} \times \{0, 1\} \\
\alpha, \beta, \cdots \in \mathcal{A} \qquad \gamma, \delta, \cdots \in \mathcal{L} & & \rho_i, \sigma_j \cdots \in \mathcal{C}
\end{array}
$$

Table 3. CaSHew-NUtS Labels

This depends on the labels defined in Table 3, which are divided into actions (α, β), on the left, and clocks (ρ, σ), on the right, and gives rise to an operational semantics in terms of a labelled transition system where terms are nodes and the edges represent behaviours labelled from the union of actions and clocks (γ), the latter being indexed from the set $0, 1$ to represent whether they respect maximal progress or not. The transition relation is of type $\mathcal{E} \times \mathcal{L} \times \mathcal{E}$, and is the greatest such relation that satisfies the rules in Table 5.

As in CaSE [8], we ensure the well-definedness of the semantics by making the negative definitions in the latter side conditions depend only on auxiliary well-formed sets (so-called initial actions, $\mathcal{IA}$, and initial clocks, $\mathcal{IC}$), rather than the transition relation itself. The main difference is in the effect of maximal progress on, and determinism of, clocks. Whereas the latter principle has an immediate preemptive effect in CaSE, *i.e.* the presence of a τ-transition removes all σ-transitions from the semantics, in CaSHew-NUtS we simply note the effect in the index to the label (*cf.* rule Com4). In particular a σ_1-labelled transition is respectful of maximal progress and deterministic, a σ_0-labelled transition is not.

Regarding determinism, whereas in CaSE the so-called 'time-out' operators $\lfloor E \rfloor \sigma(F)$ and $\lceil E \rceil \sigma(F)$, by which clocks are introduced, overrides previous transitions on that clock, CaSHew-NUtS has variant operators $[\![E]\!] \sigma(F)$ and $\llbracket E \rrbracket \sigma(F)$ where the index of the previous clock is simply decremented so that there is at most one deterministic transition, labelled σ_1, per clock σ.

We take advantage of these changes in the semantics by having two hiding operators in CaSHew-NUtS. The first, E/σ, brings hidden clocks back in line with CaSE by only turning into silent actions deterministic, maximal progress-respectful clocks, σ_1. As in CaSE, this both closes the scope of a clock, so that it is neither synchronised, nor open to preemption by, the environment, but is capable of preempting open clocks (a kind of hierarchical scoping, as explained in [8]). In order to exploit the new semantics we have a second hiding operator, $E /\!\!/ \sigma$, which allows non-deterministic and non-maximal-progress-respecting clocks to be hidden as well. This allows us to synchronise agents that still have internal work to do, as we shall later consider in giving semantics to the 'Split' operator.

In this work we shall consider only deterministic clocks so we derive the CaSE timeout operator, and several other derived operators used in that system, as follows:

$$
\begin{array}{rlrl}
\underline{a}.E = a.E + \Delta & & \lfloor E \rfloor \sigma(F) = [\![E + \Delta_\sigma]\!] \sigma(F) & \Delta_T = \Sigma_{\sigma \in T} \Delta_\sigma \\
\underline{a}_T.E = a.E + \Delta_T & & \lceil E \rceil \sigma(F) = \llbracket E + \Delta_\sigma \rrbracket \sigma(F) & \\
\underline{\sigma}.E = \lceil \mathbf{0} \rceil \sigma(E) & & \vec{a}.E = \Sigma_{i < |\vec{a}|}\, a_i.\langle a_1 \cdots a_{(i-1)} \cdot a_{(i+1)} \cdots a_{|\vec{a}|} \rangle.E & \\
\underline{\sigma}_T.E = \lceil \Delta_T \rceil \sigma(E) & & \text{where } |\vec{a}| > 1;\ \langle a \rangle.E = a.E &
\end{array}
$$

Table 4. Derived CaSEew-NUtS Syntax

$$\textsf{Idle}\ \frac{}{\mathbf{0} \xrightarrow{\sigma_1} \mathbf{0}} \qquad \textsf{Stall}\ \frac{}{\Delta_\sigma \xrightarrow{\rho_1} \Delta_\sigma}\,1$$

$$\textsf{Act}\ \frac{}{\alpha.E \xrightarrow{\alpha} E} \qquad \textsf{Patient}\ \frac{}{a.E \xrightarrow{\sigma_1} a.E}$$

$$\textsf{Sum1}\ \frac{E \xrightarrow{\alpha} E'}{E + F \xrightarrow{\alpha} E'} \qquad \textsf{Sum2}\ \frac{F \xrightarrow{\alpha} F'}{E + F \xrightarrow{\alpha} F'}$$

$$\textsf{Sum3}\ \frac{E \xrightarrow{\sigma_i} E' \quad F \xrightarrow{\sigma_j} F'}{E + F \xrightarrow{\sigma_{i \cdot j}} E' + F'}$$

$$\textsf{Com1}\ \frac{E \xrightarrow{\alpha} E'}{E \mid F \xrightarrow{\alpha} E' \mid F} \qquad \textsf{Com2}\ \frac{F \xrightarrow{\alpha} F'}{E \mid F \xrightarrow{\alpha} E \mid F'}$$

$$\textsf{Com3}\ \frac{E \xrightarrow{a} E', F \xrightarrow{\overline{a}} F'}{E \mid F \xrightarrow{\tau} E' \mid F'} \qquad \textsf{Com4}\ \frac{E \xrightarrow{\sigma_i} E' \quad F \xrightarrow{\sigma_j} F'}{E \mid F \xrightarrow{\sigma_{i \cdot j \cdot k}} E' \mid F'}\,b$$

$$\textsf{TO1}\ \frac{}{[\![E]\!]\sigma(F) \xrightarrow{\sigma_i} F}\,a \qquad \textsf{TO2}\ \frac{E \xrightarrow{\sigma_i} E'}{[\![E]\!]\sigma(F) \xrightarrow{\sigma_0} E'}$$

$$\textsf{TO3}\ \frac{E \xrightarrow{\gamma} E'}{[\![E]\!]\sigma(F) \xrightarrow{\gamma} E'}\,2$$

$$\textsf{STO1}\ \frac{}{\llbracket E \rrbracket\sigma(F) \xrightarrow{\sigma_i} F}\,a \qquad \textsf{STO2}\ \frac{E \xrightarrow{\sigma_i} E'}{\llbracket E \rrbracket\sigma(F) \xrightarrow{\sigma_0} E'}$$

$$\textsf{STO3a}\ \frac{E \xrightarrow{\alpha} E'}{\llbracket E \rrbracket\sigma(F) \xrightarrow{\alpha} E'}\,2 \qquad \textsf{STO3b}\ \frac{E \xrightarrow{\rho_i} E'}{\llbracket E \rrbracket\sigma(F) \xrightarrow{\rho_i} \llbracket E' \rrbracket\sigma(F)}\,1$$

$$\textsf{Hid1}\ \frac{E \xrightarrow{\sigma_1} E'}{E/\sigma \xrightarrow{\tau} E'/\sigma} \qquad \textsf{Hid2}\ \frac{E \xrightarrow{\alpha} E'}{E/\sigma \xrightarrow{\alpha} E'/\sigma}$$

$$\textsf{Hid3}\ \frac{E \xrightarrow{\rho_i} E'}{E/\sigma \xrightarrow{\rho_i} E'/\sigma}\,1, c$$

$$\textsf{UHid1}\ \frac{E \xrightarrow{\sigma_i} E'}{E /\!\!/ \sigma \xrightarrow{\tau} E' /\!\!/ \sigma} \qquad \textsf{UHid2}\ \frac{E \xrightarrow{\alpha} E'}{E /\!\!/ \sigma \xrightarrow{\alpha} E' /\!\!/ \sigma}$$

$$\textsf{UHid3}\ \frac{E \xrightarrow{\rho_i} E'}{E /\!\!/ \sigma \xrightarrow{\rho_i} E' /\!\!/ \sigma}\,1, d$$

$$\textsf{Res}\ \frac{E \xrightarrow{\gamma} E'}{E \setminus a \xrightarrow{\gamma} E' \setminus a}\,\gamma \notin \{a, \overline{a}\} \qquad \textsf{Rel}\ \frac{E \xrightarrow{\gamma} E'}{E[f] \xrightarrow{f(\gamma)} E'[f]}$$

$$\textsf{Rec}\ \frac{E \xrightarrow{\gamma} E'}{\mu X.E \xrightarrow{\gamma} E'\{\mu X.E/X\}}$$

where: 1) $\rho \neq \sigma$ and: a) $i = 0$ if $\tau \in \mathcal{IA}(E)$, 1 otherwise
2) $\nexists i \cdot \gamma = \sigma_i$ b) $k = 0$ if $\tau \in \mathcal{IA}(E \mid F)$, 1 otherwise
c) $\sigma_1 \notin \mathcal{IA}(E)$
d) $\nexists i \cdot \sigma_i \in \mathcal{IA}(E)$

Table 5. Operational Semantics for CaSHew-NUtS

$$
\begin{aligned}
\mathcal{IA}(\mathbf{0}) &= \emptyset & \mathcal{IA}(E+F) &= \mathcal{IA}(E) \cup \mathcal{IA}(F) \\
\mathcal{IA}(\Delta) &= \emptyset & \mathcal{IA}(E \mid F) &= \mathcal{IA}(E) \cup \mathcal{IA}(F) \\
\mathcal{IA}(\Delta_\sigma) &= \emptyset & & \quad \cup \{\tau \mid a \in \mathcal{IA}(E) \wedge \overline{a} \in \mathcal{IA}(F)\} \\
\mathcal{IA}(a.E) &= \{a\} & \mathcal{IA}(\mu X.E) &= \mathcal{IA}(E) \\
\mathcal{IA}(\tau.E) &= \{\tau\} & \mathcal{IA}(X) &= \emptyset \\
\mathcal{IA}(\lfloor E \rfloor \sigma(F)) &= \mathcal{IA}(E) & \mathcal{IA}(E \setminus L) &= \mathcal{IA}(E) \setminus (L \cup \overline{L}) \\
\mathcal{IA}(\lceil E \rceil \sigma(F)) &= \mathcal{IA}(E) & \mathcal{IA}(E/\sigma) &= \mathcal{IA}(E) \cup \{\tau \mid \sigma_1 \in \mathcal{IC}(E)\} \\
& & \mathcal{IA}(E /\!\!/ \sigma) &= \mathcal{IA}(E) \cup \{\tau \mid \sigma_i \in \mathcal{IC}(E)\}
\end{aligned}
$$

Table 6. Initial Action Set

$$
\begin{aligned}
\mathcal{IC}(\mathbf{0}) &= \{\sigma_1 \mid \sigma \in \mathcal{T}\} & \mathcal{IC}(E+F) &= \{\sigma_{i \cdot j} \mid \sigma_i \in \mathcal{IC}(E) \\
& & & \qquad \wedge\, \sigma_j \in \mathcal{IC}(F)\} \\
\mathcal{IC}(\Delta) &= \emptyset & \mathcal{IC}(E \mid F) &= \{\sigma_{i \cdot j \cdot (1 - |\{\tau\} \cap \mathcal{IA}(E \mid F)|)} \mid \\
& & & \qquad \sigma_i \in \mathcal{IC}(E) \\
\mathcal{IC}(\Delta_\sigma) &= \{\rho_1 \mid \rho \in \mathcal{T} \wedge \rho \neq \sigma\} & & \qquad \wedge\, \sigma_j \in \mathcal{IC}(F)\} \\
\mathcal{IC}(a.E) &= \{\sigma_1 \mid \sigma \in \mathcal{T}\} & \mathcal{IC}(\mu X.E) &= \mathcal{IC}(E) \\
\mathcal{IC}(\tau.E) &= \emptyset & \mathcal{IC}(X) &= \emptyset \\
\mathcal{IC}(\lfloor E \rfloor \sigma(F)) &= \mathcal{IC}(E) & \mathcal{IC}(E \setminus L) &= \mathcal{IC}(E) \\
& \quad \cup \{\sigma_1 \mid \tau \notin \mathcal{IA}(E)\} & \mathcal{IC}(E/\sigma) &= \begin{cases} \emptyset & (\text{if } \sigma_1 \in \mathcal{IC}(E)) \\ \mathcal{IC}(E) & (\text{otherwise}) \end{cases} \\
& \quad \cup \{\sigma_0 \mid \tau \in \mathcal{IA}(E)\} & & \\
\mathcal{IC}(\lceil E \rceil \sigma(F)) &= \mathcal{IC}(E) & \mathcal{IC}(E /\!\!/ \sigma) &= \begin{cases} \emptyset & (\text{if } \exists i \cdot \sigma_i \in \mathcal{IC}(E)) \\ \mathcal{IC}(E) & (\text{otherwise}) \end{cases} \\
& \quad \cup \{\sigma_1 \mid \tau \notin \mathcal{IA}(E)\} & & \\
& \quad \cup \{\sigma_0 \mid \tau \in \mathcal{IA}(E)\} & &
\end{aligned}
$$

Table 7. Initial Clock Set

A symmetric relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a *weak bisimulation* if whenever $\langle P, Q \rangle \in \mathcal{R}$:

- If $P \xrightarrow{\gamma} P'$, $\gamma \neq \tau$, then $\exists Q' \cdot Q \xrightarrow{\tau}{}^* \xrightarrow{\gamma} \xrightarrow{\tau}{}^* Q'$ and $\langle P', Q' \rangle \in \mathcal{R}$
- If $P \xrightarrow{\tau} P'$ then $\exists Q' \cdot Q \xrightarrow{\tau}{}^* Q'$ and $\langle P', Q' \rangle \in \mathcal{R}$

We say that P is *weakly equivalent* to Q and write $P \approx Q$, if $\langle P, Q \rangle \in \mathcal{R}$ for some weak bisimulation $\mathcal{R}$.

A symmetric relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a *temporal observation congruence* if whenever $\langle P, Q \rangle \in \mathcal{R}$:

1. $P \xrightarrow{\alpha} P'$ implies $\exists Q' . Q \xrightarrow{\tau}{}^* \xrightarrow{\alpha} \xrightarrow{\tau}{}^* Q'$ and $P' \approx Q'$.
2. $P \xrightarrow{\sigma} P'$ implies $\exists Q' . Q \xrightarrow{\sigma} Q'$ and $\langle P', Q' \rangle \in \mathcal{R}$.

Proposition 1. *Compositionality*
Temporal observation congruence is compositional through all operators.

Proposition 2. *Full Abstraction*
Temporal observation congruence is the coarsest congruence contained in temporal weak bisimulation.

4 CASheW-S Semantics in CaSHew-NUtS

In the semantic translation from the CASheW-S to the CaSHew-NUtS language we will use the variables in Table 8.

p	: Process	q	: Performance
m	: Process Name	n, o	: Performance Name
a, b	: (Process) Input	a_j^n, b_j^o	: (Performance) Input Component
c, d	: (Process) Output	c^n, d^o	: (Performance) Broadcast Output
g	: Consume	u	: Value Data
h	: Produce	v	: Value Collector
w	: AProcess	z	: CProcess

Table 8. Variables

When we want to represent a collection we will use the corresponding capital, for instance A is a set of inputs; we abuse this syntax slightly by allowing lists to be represented the same way so that G represents, for instance, a consume list as defined in the CASheW-S syntax. Finally Q stretches the notation further by representing a CASheW-S performance list which has connections, as well as performances, as members and strictly has a performance at the head (and is expanded by '*snoc*ing', *i.e.* adding new members to the tail, and decomposed by reverse tail recursion).
At the top level we look at the semantics of processes. Our main semantic function $[\![]\!]$ is of type p –> m –> A –> C –> $\mathcal{E}$ (which then composes with the CaSHew-NUtS semantic function to derive a labelled transition system). There are two possibilities matched by this function: the atomic process and the composite process.

$$
\begin{aligned}
&{}^m[\![\textbf{AtomicProcess } m\ w]\!]^A_C = {}^m[\![w]\!]^A_C \\
&{}^m[\![\textbf{CompositeProcess } m\ z\ G\ H]\!]^A_C \\
&\qquad = ({}^m[\![z]\!]^{A^m}_{C^m} \mid [\![G]\!]^A_\emptyset \mid [\![H]\!]^\emptyset_C) \setminus A^m \cup C^m \,/\, \{\sigma^{c^n} \mid c^n \in C^m\}
\end{aligned}
$$

Table 9. Process Semantics

For an atomic process the process name must match the one declared syntactically, the inputs and outputs must match those semantically associated with the AProcess, the syntactic nature of which we have left open. One possibility is to consider all services as grounded in WSDL and having 'functional' behaviour, *i.e.* with all inputs required, and all outputs produced, at every execution. In this case all AProcess semantics will take the form shown in Figure 1, generalised for different numbers of inputs and outputs, and the match will be syntactically based on a representation of WSDL. Our aim in the wider context, however, is to plug in a semantic translation of a choreography language at this point.

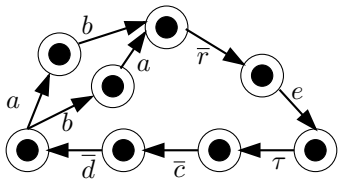

Fig. 1. Example AProcess Semantics

WSMO has proposed that two process models should be associated with a service to be properly described [11]. One, which we concentrate on in this paper, is named the *orchestration* and focuses on how the behaviour of a composite process is formed from the behaviour of its component services. The *choreography*, on the other hand establishes, in terms of interaction, how a client should interact with the service. There are many forms that this could take, and as yet little agreement, but our own previous work [9] has generalised on the ideas of *interface automata* [2]. We previously concentrated on data-driven scheduling of generalised dataflow-oriented systems, and showed how to accommodate statefulness, optional inputs and non-determinism in semantics such a scheme. In particular, automata allow us to easily mix statefulness and non-determinism, by representing the internal behaviour with an explicit silent action τ or choice between these. To these transition labels for inputs and outputs (we overline outputs as is usual in CCS) we add *'scheduling signals'* that allow us to be explicit about *'readiness'* for execution. The signal r signifies readiness (and can be non-deterministically offered alongside further inputs to show that these are optional), and is followed by signal e, which signifies permission to execute.

All of these features are widely claimed necessary in the composition of semantic web services, where the notion of service is as much based on work in agents as on SOAP/WSDL web services. The Any-Order composite process is given informal semantics as interleaving execution of the components explicitly according to their readiness to execute (based on inputs as well as non-data preconditions, from which we abstract).

The guiding principle for our semantics will therefore be drawn from our existing model scheme [8], where the prototypical level (here called processes, there components) are described in terms of such automata, and where compositions at the instance level (here called performances, there component instances) will be given a compositional semantics by means of a 'token passing game', synchronised by clocks. The ability to turn such clocks into silent actions, away from which we can abstract away in our equivalence theory in temporal observation congruence, gives us a means to form such an interface automaton (with no explicit clocks) for a composite process. As shown, the inputs and outputs for such are based on the Produces and Consumes contained, the other inputs and outputs of the composite CProcess being restricted away. At the same time, the clocks that coordinate the outputs, as described later, are hidden according to maximal progress.

$$[\![\textbf{Consume } a\ n\ b\ j]\!]^{\{a\}}_{\emptyset} = \mu X.a.\overline{\underline{b^n_j}}.X$$
$$[\![\textbf{Produce } c\ n\ d]\!]^{\emptyset}_{\{c\}} = \mu X.\underline{d^n}.\overline{c}.X$$

Table 10. Produce and Consume Semantics

Table 10 shows the semantics that are given to Consume and Produce declarations. In basic terms, these cyclically convert from process inputs to performance inputs, and from performance outputs to process outputs, respectively. The underlining in the syntax, as defined in the derived syntax, represents the timing of the two communications involved: the initial input is *'patient'*, meaning that an unspecified amount of time can pass on all clocks (*cf.* rule Patient) while the agent waits for the input; the subsequent output is *'insistent'* meaning that this communication is instantaneous, *i.e.* can be measured on no clock. This is represented in transition diagrams for these two agents shown in Figures 2 and 3 respectively. The double circle means that any clock not explicitly shown *'idles'*, *i.e.* has a self-transition at the state; the single circle means that any clock not explicitly shown cannot tick, *i.e.* has no transitions.

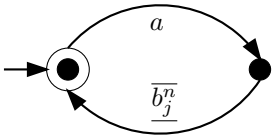

Fig. 2. Consume Semantics

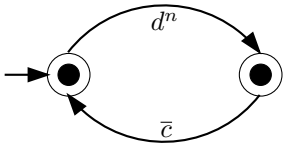

Fig. 3. Produce Semantics

These agents are composed, to make a ConsumeList and ProduceList respectively, according to the general composition semantics shown in Table 11 (where x can stand for any symbol, including the absence of any such, and K and L any non-bracketed list, *i.e.* any list in the CAShe-W-S syntax except the PerformanceList). This is based directly on parallel composition and the accumulation of inputs and outputs.

$${}^{x}[\![K; L]\!]^{A^L \cup A^K}_{C^K \cup C^L} = {}^{x}[\![K]\!]^{A^K}_{C^K} \mid {}^{x}[\![L]\!]^{A^L}_{C^L}$$

Table 11. General Composition Semantics

This composition is also used in forming ValueDataLists and ValueCollectorLists from the individual semantics shown in Table 12.

$$[\![\textbf{ValueData } a]\!]_{\{a\}} = \mu X.\overline{a}.X$$
$${}^{n}[\![\textbf{ValueCollector } a\ k]\!]^{\{a^n_j \mid j<k\}}_{a} = \mu X.\langle a^n_j \mid j < k\rangle.\tau.\overline{\underline{a}}.X$$

Table 12. Data Aggregation Semantics

Using this we are able to form semantics for performances as shown in Table 13.

$$
\begin{aligned}
&{}^{(m,n)}[\![\textbf{Perform } n\ p\ U\ V]\!]^{\{a_0^n \mid a \in A^m \wedge a \notin C^U \wedge a \notin C^V\} \cup A^V}_{\{c^n \mid c \in C^m\}} \\
&\quad = ([\![U]\!]_{C^U} \mid [\![V]\!]^{A^V}_{C^V} \mid {}^m[\![p]\!]^{A^m}_{C^m}[\{a \mapsto a_0^n \mid a \in A^m \wedge a \notin C^U \wedge a \notin C^V\}] \\
&\qquad \mid \Pi_{c \in C^m} \mu X.\underline{c}_{\sigma^{c^n}}.\lceil \mu Y.\underline{\overline{c^n}}_{\sigma^m}.Y \rceil \sigma^{c^n}(X)) \setminus C^U \cup C^V \cup C^m
\end{aligned}
$$

Table 13. Performance Semantics

This composes the semantics of the ValueDataList and ValueCollectorList with the process being performed, having renamed those inputs not removed by a ValueData or componentised by a ValueCollector to form a single-component performance input (a_0^n), with one agent per process output (Π represents distributed parallel composition) that turns these into broadcast outputs. This broadcast agent is illustrated with the transition diagram in Figure 4. While waiting for a value this is patient in all clocks but is instantaneous in the unique associated clock σ^{c^n}. Once the process output c is received, it will be broadcast as c^n until the associated clock ticks. As we know from the semantics for composite processes, each such clock will be hidden under the conditions of maximal progress. This means that whenever there is an agent that can receive the broadcast, the subsequent silent action will prevent the clock. In this way the instant measured by the clock will necessarily contain each such communication.
We arrange for this communication by giving connections the semantics detailed in Table 14 and illustrated in Figure 5.

$$
{}^m[\![\textbf{Connect } n\ c\ o\ a\ j]\!] = \mu X.\, c^n.\, \underline{\overline{a_j^o}}.\sigma^{c^n}.\, X
$$

Table 14. Connection Semantics

This agent patiently waits for the broadcast but then insistently relays this to the recipient performance. Only the value has been passed on will the agent synchronise on the associated clock to signal the end of the broadcast instant. It must wait for this clock before picking up a new value to avoid duplicates.

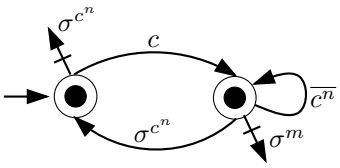

Fig. 4. Broadcast Semantics

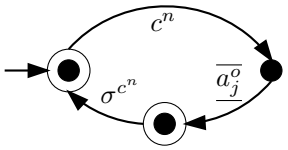

Fig. 5. Connection Semantics

Having shown the semantics for performances we are now able to continue the semantics for composite processes. At the top level the semantics for CProcess are as shown in Table 15.

$$
\begin{aligned}
{}^{m}[\![\textbf{AnyOrder}\ Q]\!]^{A}_{C} &= {}^{m}_{an}[\![Q]\!]^{A}_{C} \setminus t \,/\, \sigma^{m} \\
{}^{m}[\![\textbf{Sequence}\ Q]\!]^{A}_{C} &= {}^{m}_{se}[\![Q]\!]^{A}_{C} \setminus t \,/\, \sigma^{m} \\
{}^{m}[\![\textbf{Split}\ Q]\!]^{A}_{C} &= ({}^{m}_{sp}[\![Q]\!]^{A}_{C} \mid \mu X.\sigma^{m}.\overline{r}.e.\sigma^{m}.\sigma^{m}.X) /\!\!/ \sigma^{m} \\
{}^{m}[\![\textbf{SplitJoin}\ Q]\!]^{A}_{C} &= ({}^{m}_{sj}[\![Q]\!]^{A}_{C} \mid \mu X.\sigma^{m}.\overline{r}.e.\sigma^{m}.\sigma^{m}.X) /\!\!/ \sigma^{m} \\
{}^{m}[\![\textbf{ChooseOne}\ Q]\!]^{A}_{C} &= ({}^{m}_{co}[\![Q]\!]^{A}_{C} \mid \mu X.\underline{r^{i}.\overline{r}.e.\overline{e^{i}}}_{\sigma^{m}}.\sigma^{m}.X) \setminus \{e^{i}, r^{i}\} \,/\, \sigma^{m} \\
{}^{m}[\![\textbf{IfThenElse}\ \ \textbf{Perform}\ n\ p^{n}\ U^{n}\ V^{n}\ \ \textbf{Perform}\ o\ p^{o}\ U^{o}\ V^{o}]\!]^{A^{n}\cup A^{o}}_{C^{n}\cup D^{o}} & \\
= ({}^{(m,n)}[\![\textbf{Perform}\ n\ p^{n}\ U^{n}\ V^{n}]\!]^{A^{n}}_{C^{n}}[e \mapsto e^{n}, r \mapsto r^{n}] \mid & \\
{}^{(m,o)}[\![\textbf{Perform}\ o\ p^{o}\ U^{o}\ V^{o}]\!]^{A^{o}}_{C^{o}}[e \mapsto e^{o}, r \mapsto r^{o}] \mid & \\
\mu X.(\tau.\underline{r^{n}.\overline{r}.e.\overline{e^{n}}}_{\sigma^{n}}.\sigma^{n}.X \ + \tau.\underline{r^{o}.\overline{r}.e.\overline{e^{o}}}_{\sigma^{o}}.\sigma^{o}.X)) & \\
\setminus \{e^{n}, e^{o}, r^{n}, r^{o}\} \,/\, \sigma^{n} \,/\, \sigma^{o} & \\
{}^{m}[\![\textbf{RepeatWhile Perform}\ n\ p\ U\ V]\!]^{A}_{C} & \\
= ({}^{(m,n)}[\![\textbf{Perform}\ n\ p\ U\ V]\!]^{A}_{C}[e \mapsto e^{i}, r \mapsto r^{i}] \mid & \\
\mu X.(\tau.\underline{r^{i}.\overline{r}.e.\overline{e^{i}}}_{\sigma^{n}}.\sigma^{n}.\mu Y.(\tau.X + \tau.\underline{r^{i}.\overline{e^{i}}}_{\sigma^{n}}.\sigma^{n}.Y) & \\
+\tau.\overline{r}.e.X)) \setminus \{e^{i}, r^{i}\} \,/\, \sigma^{n} & \\
{}^{m}[\![\textbf{RepeatUntil Perform}\ n\ p\ U\ V]\!]^{A}_{C} & \\
= ({}^{(m,n)}[\![\textbf{Perform}\ n\ p\ U\ V]\!]^{A}_{C}[e \mapsto e^{i}, r \mapsto r^{i}] \mid & \\
\mu X.\underline{r^{i}.\overline{r}.e.\overline{e^{i}}}_{\sigma^{n}}.\sigma^{n}.\mu Y.(\tau.X + & \\
\tau.\underline{r^{i}.\overline{e^{i}}}_{\sigma^{n}}.\sigma^{n}.Y)) \setminus \{e^{i}, r^{i}\} \,/\, \sigma^{n} &
\end{aligned}
$$

Table 15. CProcess Semantics

The first five types of composite process — Any-Order, Sequence, Split, Split-Join and ChooseOne — are defined over a list of performances and connections and we should therefore like to form a semantics for the list which is open to further composition under the clock σ^{m}. Since the exact form of composition depends on the process context, we define a family of semantic functions where this is a parameter. These functions are detailed in Table 17. Since the list may also include connections, we include a generic rule for composing these in Table 16.

$$
{}^{m}_{x}[\![(Q); \textbf{Connect}\ n\ c\ o\ a\ j]\!]^{A}_{C} = {}^{m}[\![\textbf{Connect}\ n\ c\ o\ a\ j]\!]^{A}_{C} \mid {}^{m}_{x}[\![Q]\!]^{A}_{C}
$$

Table 16. Connection Composition Semantics

The other types of composition are expressed directly in terms of their components and we similarly give them direct semantics, using silent transitions to encode the non-deterministic choice that is implicit.

$$
\begin{aligned}
&{}^{m}_{an}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A}_{C} \\
&\quad = ({}^{(m,n)}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A}_{C}[e \mapsto e^i, r \mapsto r^i] \mid \\
&\quad\quad \mu X.\underline{r^i}_{\{\sigma^m,\sigma^n\}}.(\underline{\overline{r}.e.\overline{e^i}}_{\{\sigma^m,\sigma^n\}}.\underline{\sigma^n}_{\sigma^m}.\lfloor\underline{\overline{t}.\sigma^m}_{\sigma^n}.X\rfloor\sigma^m(X) + \\
&\quad\quad \underline{t.\overline{e^i}.\sigma^n}_{\sigma^m}.\lfloor\underline{\overline{t}.\sigma^m}_{\sigma^n}.X\rfloor\sigma^m(X))) \setminus \{e^i, r^i\}/\sigma^n \\
&{}^{m}_{an}[\![(Q); \textbf{Perform } n\ p\ U\ V]\!]^{A^Q \cup A^n}_{C^Q \cup C^n} \\
&\quad = {}^{m}_{an}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A^n}_{C^n} \mid {}^{m}_{an}[\![Q]\!]^{A^Q}_{C^Q}
\end{aligned}
$$

$$
\begin{aligned}
&{}^{m}_{se}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A}_{C} \\
&\quad = ({}^{(m,n)}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A^n}_{C^n}[e \mapsto e^i, r \mapsto r^i] \mid \\
&\quad\quad \mu X.\underline{r^i.\overline{r}.e.\overline{e^i}}_{\{\sigma^m,\sigma^n\}}.\underline{\sigma^n}_{\sigma^m}\lfloor\underline{\overline{t}.\sigma^m}_{\sigma^n}.X\rfloor\sigma^m(X))/\sigma^n \setminus \{r^i, e^i\} \\
&{}^{m}_{se}[\![(Q); \textbf{Perform } n\ p\ U\ V]\!]^{A^Q \cup A^n}_{C^Q \cup C^n} \\
&\quad = ({}^{n}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A^n}_{C^n}[e \mapsto e^i, r \mapsto r^i] \mid {}^{m}_{se}[\![Q]\!]^{A^Q}_{C^Q}[t \mapsto t^i] \mid \\
&\quad\quad \mu X.\underline{t^i}_{\sigma^n}.\underline{r^i.\overline{e^i}}_{\{\sigma^m,\sigma^n\}}.\underline{\sigma^n}_{\sigma^m}\lfloor\underline{\overline{t}.\sigma^m}_{\sigma^n}.X\rfloor\sigma^m(X))/\sigma^n \setminus \{r^i, e^i\}
\end{aligned}
$$

$$
\begin{aligned}
&{}^{m}_{sp}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A}_{C} \\
&\quad = ({}^{(m,n)}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A}_{C}[e \mapsto e^i, r \mapsto r^i] \mid \\
&\quad\quad \mu X.\underline{r^i}_{\{\sigma^m,\sigma^n\}}.\underline{\sigma^m.\sigma^m}_{\sigma^n}.\underline{\overline{e^i}}_{\{\sigma^m,\sigma^n\}}.\underline{\sigma^m}_{\sigma^n}.X) \setminus \{e^i, r^i\}/\sigma^n \\
&{}^{m}_{sp}[\![(Q); \textbf{Perform } n\ p\ U\ V]\!]^{A^Q \cup A^n}_{C^Q \cup C^n} \\
&\quad = {}^{m}_{sp}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A^n}_{C^n} \mid {}^{m}_{sp}[\![Q]\!]^{A^Q}_{C^Q}
\end{aligned}
$$

$$
\begin{aligned}
&{}^{m}_{sj}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A}_{C} \\
&\quad = ({}^{(m,n)}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A}_{C}[e \mapsto e^i, r \mapsto r^i] \mid \\
&\quad\quad \mu X.\underline{r^i}_{\{\sigma^m,\sigma^n\}}.\underline{\sigma^m.\sigma^m}_{\sigma^n}.\underline{\overline{e^i}}_{\{\sigma^m,\sigma^n\}}.\underline{\sigma^n}_{\sigma^m}.\underline{\sigma^m}_{\sigma^n}.X) \setminus \{e^i, r^i\}/\sigma^n \\
&{}^{m}_{sj}[\![(Q); \textbf{Perform } n\ p\ U\ V]\!]^{A^Q \cup A^n}_{C^Q \cup C^n} \\
&\quad = {}^{m}_{sj}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A^n}_{C^n} \mid {}^{m}_{sj}[\![Q]\!]^{A^Q}_{C^Q}
\end{aligned}
$$

$$
\begin{aligned}
&{}^{m}_{co}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A}_{C} \\
&\quad = {}^{(m,n)}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A}_{C}[e \mapsto e^i, r \mapsto r^i]/\sigma^n \\
&{}^{m}_{co}[\![(Q); \textbf{Perform } n\ p\ U\ V]\!]^{A^Q \cup A^n}_{C^Q \cup C^n} \\
&\quad = {}^{m}_{co}[\![\textbf{Perform } n\ p\ U\ V]\!]^{A^n}_{C^n} \mid {}^{m}_{co}[\![Q]\!]^{A^Q}_{C^Q}
\end{aligned}
$$

Table 17. Performance Composition Semantics

5 Related Work

The original semantics for DAML-S were provided via translation to Petri Nets [7]. As well as problems with providing compositionality for a mathematical semantics for these, the translation was fundamentally non-compositional. Synchronisations were built for the fixed number of performances involved, for each form of composite behaviour, that are not open to the composition of further performances. Furthermore, the semantics was provided for a very early version of DAML-S, the forerunner to OWL-S, where only control flow and no data flow was described. The question of the effect of data on control flow, which we have modelled as an explicit 'readiness to execute' signal, was therefore not considered at all. This would very much restrict the ability to use that model for analysis.

A more developed operational semantics have been provided in process calculus-like style derived from Concurrent Haskell/Erlang semantics [1]. In this work an intermediate language called *'Core DAML-S'* is treated to structured operational semantics like shown here for CaSHew-NUtS. Unfortunately no compositionality result is provided, or provable, for the Core DAML-S semantics since no equivalence theory is nominated. Furthermore, again the translation from the full process model is non-compositional since fixed size 'spawn' processes are created, as are agents which wait for fixed numbers of synchronisations signalling completion, not open to further composition once formed. Finally, since the dataflow for loop-type processes were not fixed, no semantics were given for these. In our formalism it is feasible nevertheless to offer semantics for the control-flow part of these processes.

6 Conclusions and Future Work

Our intention in establishing compositional operational semantics for OWL-S is twofold. First we should like to implement the semantics to provide an orchestration engine, which we are developing as an open source project in Haskell. The previous semantics have inspired the so-called *DAML-S Virtual Machine* [10], though this has not been made widely available. An informal argument about correctness of the DAML-S Virtual Machine is indirect, based on re-interpretation of the semantics as logical predicates. Our implementation will be more direct, with an inductive datatype directly representing the CaSHew-NUtS syntax and a step function directly representing its operational semantics.

Our second aim is to extend the verification results we have for our previous model [8]. In particular we should like to check the consistency of dataflows in an automatic fashion. In the same way as this has been cast as a system of behavioural types in our previous work [9], we should like to establish a formal link between orchestration and choreography. Whereas these are seperate models in current approaches, our belief is that application developers using service-oriented architectures should assign choreography models at each level of composition and a formal check that this choreography is consistent with the orchestration defined should implicitly check the internal consistency of the orchestration.

References

1. Anupriya Ankolekar, Frank Huch, and Katia Sycara. Concurrent execution semantics of DAML-S with subtypes. In *Proc. 1st Intl. Semantic Web Conference (ISWC2002)*, volume 2342 of *LNCS*, pages 308–332. Springer Verlag, May 2002.
2. L. de Alfaro and T.A. Henzinger. Interface automata. In *Proc. 8th European Soft. Eng. Conference and 9th ACM SIGSOFT International Symposium on Foundations of Soft. Eng. (ESEC/FSE 2001)*, volume 26, 5 of *Software Engineering Notes*, pages 109–120. ACM Press, 2001.
3. David Martin *et al.* OWL-S: Semantic markup for web services. http://www.daml.org/services/owl-s/1.1/overview/, 2004.
4. M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221–239, March 1995.
5. A. J. R. G. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
6. A. J. R. G. Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambrudge University Press, 1999.
7. Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *Proc. 11th Intl. World Wide Web Conference (WWW2002)*, May 7-10 2002.
8. B. Norton, G. Lüttgen, and M. Mendler. A compositional semantic theory for synchronous component-based design. In *14th Intl. Conference on Concurreny Theory (CONCUR '03)*, number 2761 in LNCS. Springer-Verlag, 2003.
9. Barry Norton and Matt Fairtlough. Reactive types for dataflow-oriented software architectures. In Danielle C. Martin, editor, *Proceedings of 4th IEEE/IFIP Conference on Software Architecture (WICSA2004)*, volume P2172, pages 211–220. IEEE Computer Society Press, 2004.
10. Massimo Paolucci, Anupriya Ankolekar, Naveen Srinivasan, and Katia Sycara. The DAML-S virtual machine. In *Proc. 2nd Intl. Semantic Web Conference (ISWC2002)*, volume 2870 of *LNCS*, pages 290–305. Springer Verlag, 2003.
11. Dumitru Roman, Holger Lausen, and Uwe Keller. WSMO final draft. http://www.wsmo.org/TR/d2/v1.1/, February 2005.
12. A. Sheth, C. Ramakrishnan, and C. Thomas. Semantics for the Semantic Web: The implicit, the formal and the powerful. *Intl. Journal on Semantic Web and Information Systems*, 1(1):1–18, 2005.

Acknowledgements

This work was carried out within the DIP project, an Integrated Project (no. FP6 - 507483) supported by the European Union's IST programme, and the Dot.Kom project, also sponsored within the IST programme (no. IST-2001-34038).