Composition of Server Processes

We want to describe server behaviour. For modular specification of server behaviour, compositions can be useful. We primarily target execution, with additional uses to follow.

Whether the process is used from the client or server perspective should not matter for the formalisation. Or rather: what is the distinction? C.f. ConsumedThing vs. ExposedThing in Web of Things Scripting API.

Representing the behaviour of a single resource (atomic) processes on the server is rather simple. But we typically need more complex behaviour involving multiple processes.

Two difficulties in a resource-oriented view:

- side effects between resources (with "hierarchy"?)
- resource creation and deletion (with "hierarchy"?)

Terminology

Modularity does not only mean hierarchy (abstraction), but also inclusion (of components?) and union (of fragments?). We talk about processes, components and fragments.

An atomic process describes the behaviour of a singleton resource (one "port"). Components and fragments are atomic. A composite process describes the behaviour of a collection of resources (many "ports").

Components have an initial state, fragments do not have an initial state.

The READ operation is always at the process or component (and not on the fragment)?

We can have:

- sequential composition: first process A, then process B. (With shared states as connection points? Or via the ε transition?)
- parallel composition: process/component/fragment A and process/component /fragment B in parallel (via shared transitions?)
- hierarchical composition: process/component/fragment A and process/component /fragment B in parallel (via a state in A that is extended with B)

With hierarchy, we have the superstate and the substates. Process (state machine) B refines process (state machine) A. B is a more detailed description of A. A is an abstraction of B. Each process has at most one super process and can have multiple sub processes. Do we need subs and super processes, or are just associated processes enough?

Currently we have a 1:1 between super and sub. An extension could be to have 1:n.

For including component processes, we do not have a hierarchy, but something more like transclusion ("The inclusion of part of hypertext document in another one by means of reference rather than copying."). Or "overlay" or "embed". How are "orthogonal regions" from UML state machines related? Or "parent" and "child", "up" and "down", "container" and "component"?

The process (state machine) /rsg/index#p is a composed (or composite) process; /ready /index#p, /set/index#p, /go/index#p are atomic processes (state machines) and are components of /rsg/index#p.

Need to represent side effects between occurrences /rsg/{id}#o and /ready/{id}#o, /set/{id}#o, /go/{id}#o, related to the resource lifecycle.

Connecting Processes

Connections

- shared transitions (parallel composition)
- expand one parent state to entire sub-process (hierarchical composition)
- use states as connection points (sequential composition)

Other

- preconditions (guards?) for transitions can be read directly from the atomic state machines
- use LTL formulas to relate states in the parent automata to the sub automata?
- Is component r (s, g) included in composition rsg?
- Is fragment reinit included in composition rsg?

Composition Semantics

Constraints and mappings?

Input: fragment state machines, component state machines, constraints (@@@what are constraints?)

Output: composite state machine

```
fragment 1 ∪ fragment 2 ∪ ... ∪ fragment n || components and constraints
```

Union? Product? ||?

Serial Composition

○

Serial composition involves connecting state machines or processes sequentially, where the output of one becomes the input of the next. The system progresses through each

component in order.

Cascade Composition

Cascade composition is similar to serial composition but allows for more complex interactions. In a cascade, the output of one component influences the behavior of the next, but the second component may still have its own inputs and outputs.

Parallel Composition

||

Parallel composition involves running two or more state machines or processes simultaneously. Each component operates independently but can synchronize on shared events or communications. Parallel composition is often denoted by the || operator.

Of server machines: synchronise on common transitions.

Of server and client machines: check compatibility (communication safety).

Interleaving Composition

Similar to parallel composition, but events from different components are interleaved rather than occurring simultaneously.

Side-by-side Composition

Side-by-side composition is essentially another term for parallel composition. Side-by-side composition emphasises the visual representation of placing state machines next to each other, operating concurrently but potentially interacting.

Hierarchical Composition

Hierarchical composition involves nesting a process within a state of another process, creating a hierarchy of behaviors. Hierarchical composition is useful for modelling systems with different levels of abstraction.

Operations/Test Cases

Given a composition of server processes, output the synchronisation points.

Given a composition of server processes, output the hierarchy (but all we have is bidirectional links/associations)? States influence each other both ways?

Reachability

Given P0 || P1 || P2 || P3, check reachability of all states.

Output states with the same incoming transition as conjunction.

Dependency Graph

Generate the dependency graph on shared transitions.

Subordinate resources are created when the parent resource is created?

Request/Response

Check that every request leads to a specified response.

Notes

Equality on DFA is decidable. Let M be a state machine; we write L(M) to denote the language of M. Given state machines (could be NFA) M*1 and M*2, L(M*1) = L(M*2), first convert NFA to DFA

https://www.youtube.com/watch?v=f11OJAPOtzY

https://www.youtube.com/@marcodinatale2201