For Coordination, State Component Transitions

Simon Bliudze
Anastasia Mavridou
Alina Zolotukhina
Rigorous System Design Laboratory (EPFL)
{firstname.surname}@epfl.ch

Radoslaw Szymanek
Crossing-Tech S.A.
radoslaw.szymanek@crossing-tech.com

With partial support from
the Swiss Commission for Technology and Innovation

Need for coordination

Independent software entities share access to resources.

Communication and data exchange can be complex.

Component execution must be coordinated.

Semaphores, locks, monitors, etc.

Coordination based on low-level primitives rapidly becomes unpractical.

Synchronisation

Task 1:

```
...
free(S1);
take(S2);
...
```

Task 2:

```
...
take(S1);
free(S2);
...
```

A simple synchronisation barrier

Synchronisation

Task 1:

```
...
free(S1);
free(S1);
take(S2);
take(S3);
...
```

Task 2:

```
...
take(S1);
free(S2);
free(S2);
take(S3);
...
```

Task 3:

```
...
take(S1);
take(S2);
free(S3);
free(S3);
...
```

Three-way synchronisation barrier

Synchronisation with data transfer

Task 1:	Task 2:
`initialise(x);`	`initialise(y);`
`free(S1);`	`take(S1);`
`take(S2);`	`free(S2);`
`sh = max(x, sh);`	`sh = max(y, sh);`
`free(S2);`	`take(S2);`
`take(S1);`	`free(S1);`
`x = sh;`	`y = sh;`

Coordination mechanisms mixed up with computation do not scale.

Synchronisation with data transfer

Task 1:

```
initialise(x);
free(S1);
take(S2);
sh = max(x, sh);
free(S2);
take(S1);
x = sh;
```

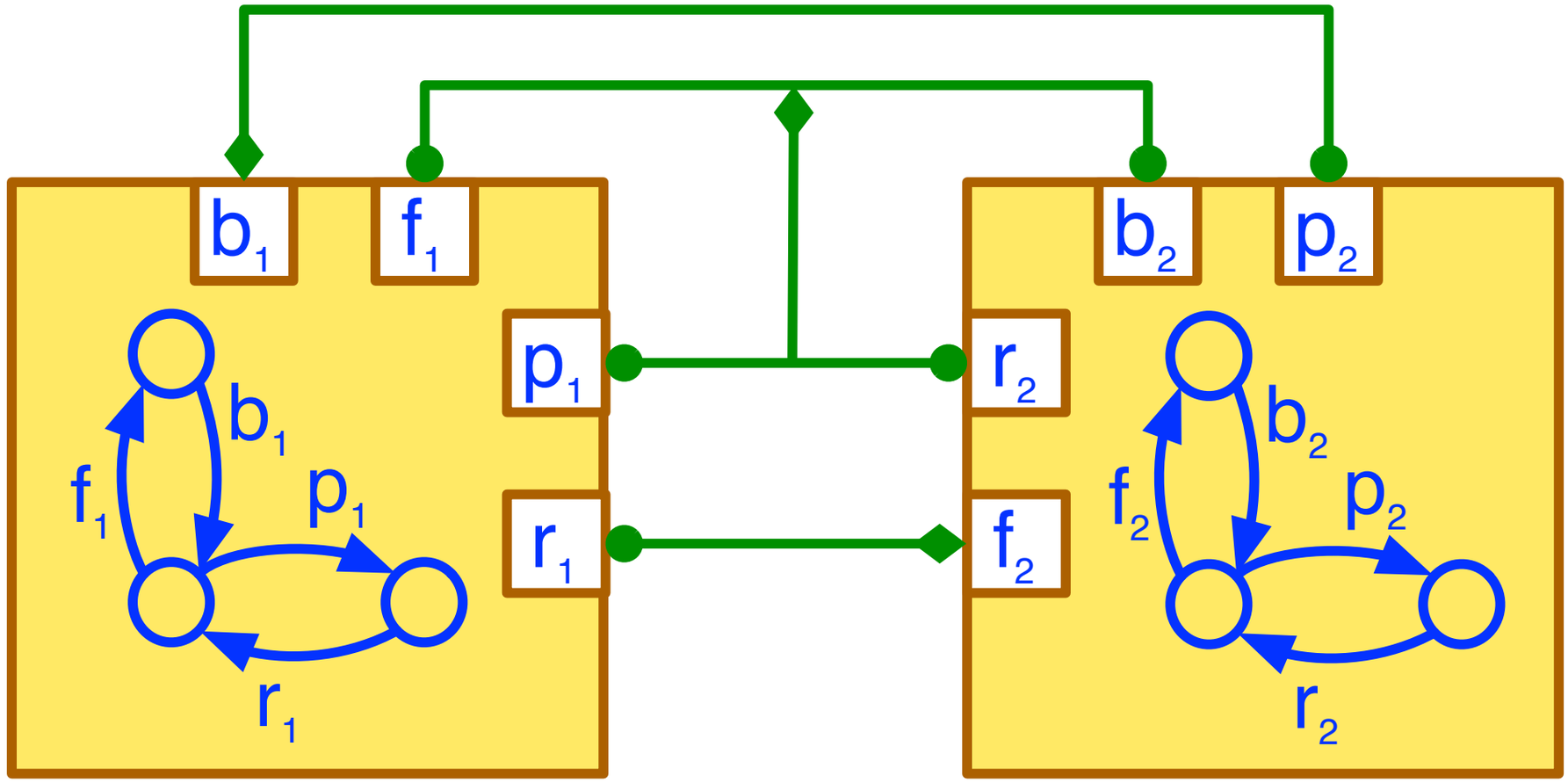
Task 2:

```
initialise(y);
take(S1);
free(S2);
sh = max(y, sh);
take(S2);
free(S1);
y = sh;
```

Coordination mechanisms mixed up with computation do not scale.

Code maintenance is a nightmare!

Separation of concerns: The BIP approach



Coordination glue is a separate entity

Component behaviour specified by Finite State Machines

BIP background

Initially developed for embedded systems control

Three layers:

- Component behaviour
- Coordination glue
- Data transfer

Glue can be synthesised and analysed for safety

- Analysis of synchronisation deadlocks

  S. Bensalem, M. Bozga, J. Sifakis, T.-H. Nguyen.
  DFinder: A Tool for Compositional Deadlock Detection and Verification [CAV'09]

- Synthesis of glue for safety properties

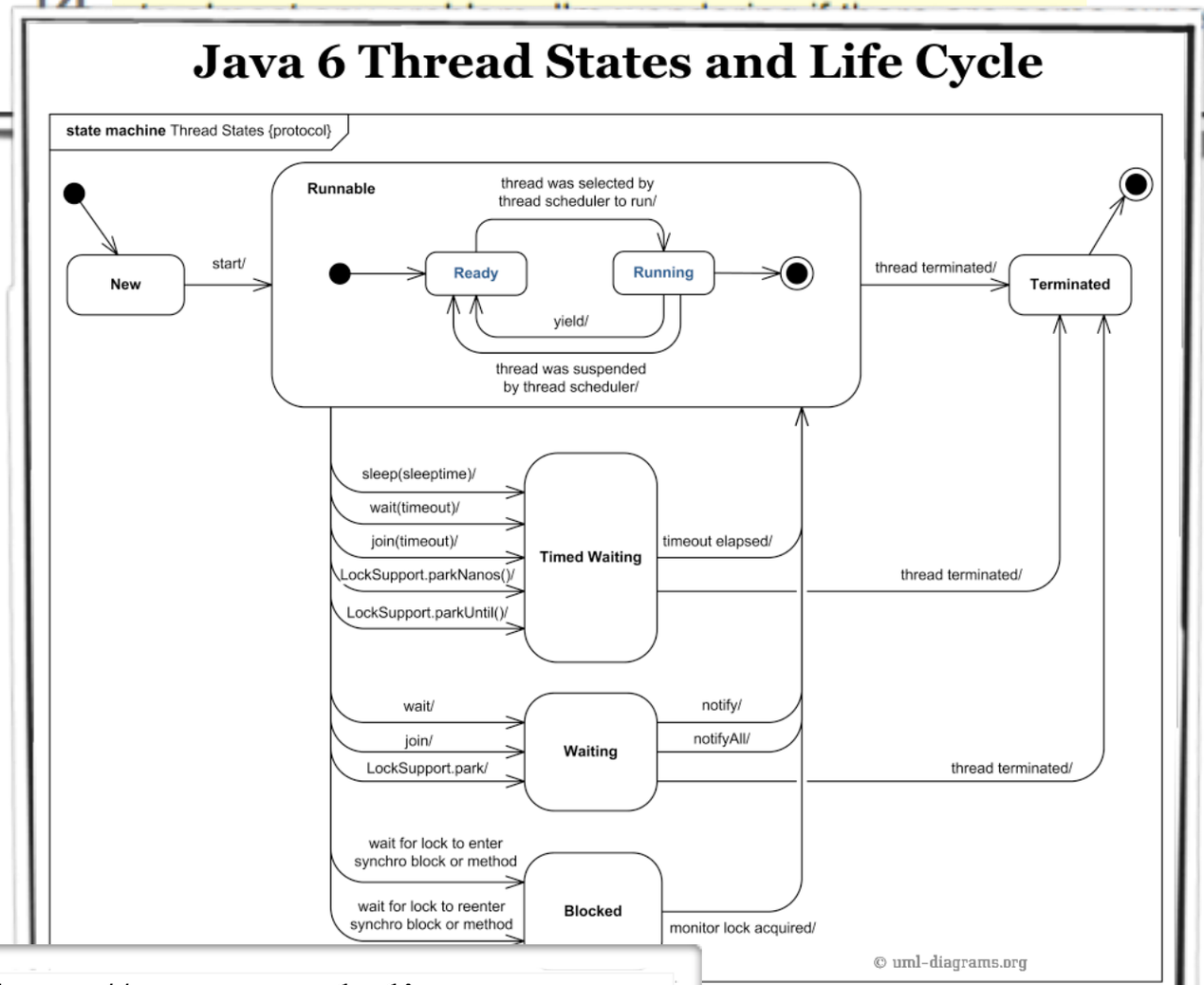
  S. Bliudze and J. Sifakis.
  Synthesizing Glue Operators from Glue Constraints for the Construction of Component-Based Systems [SC'11]

Finite State Machine — A good abstraction

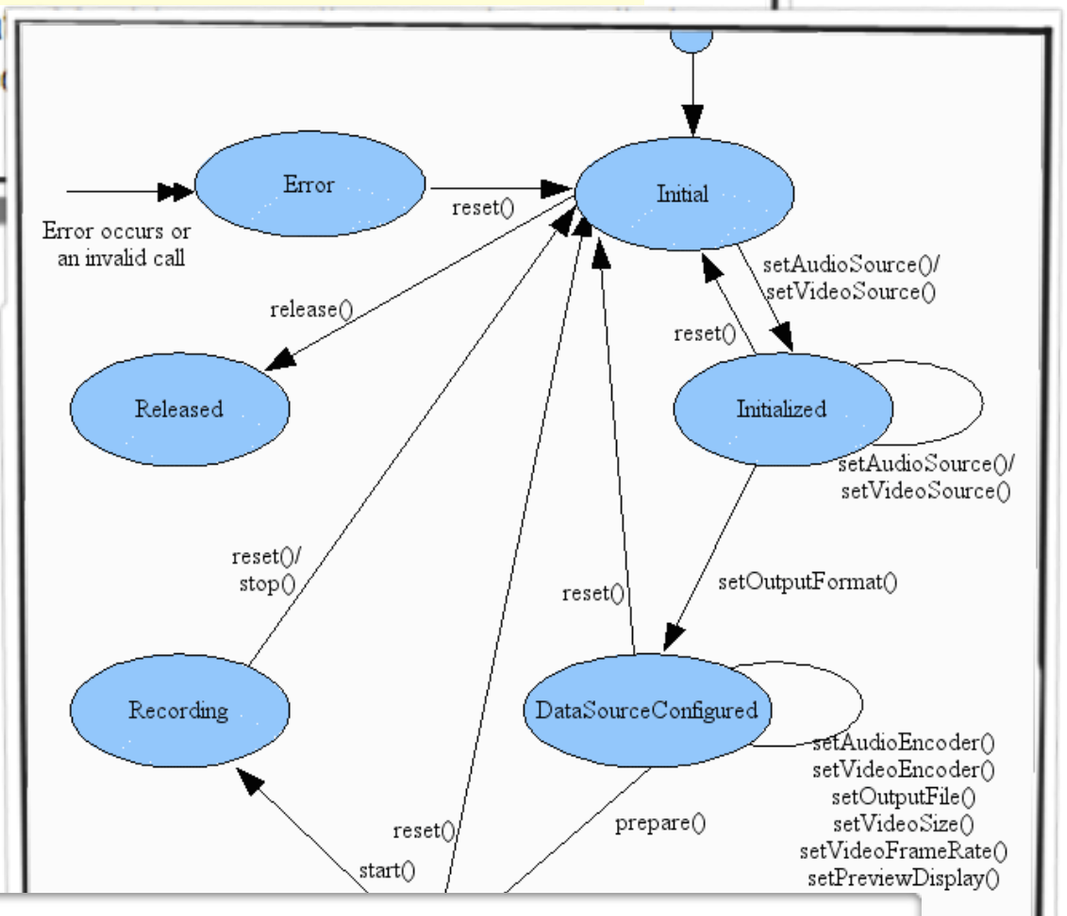
stackoverflow Questions Tags Tour Users

State Machines and User Interface work — any examples/experience?

I'm looking for ways to de-spaghttify my front-end widget code. It's been suggested that a Finite State Machine is the right way to think about what I'm doing. I know a Sta

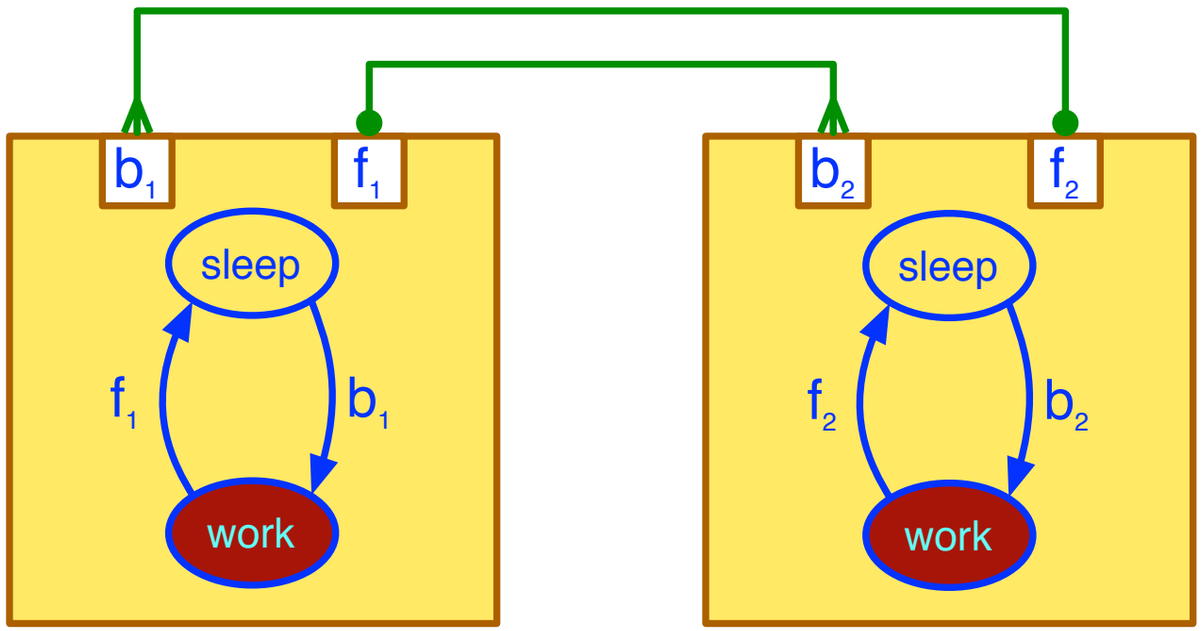


http://www.uml-diagrams.org



Android MediaRecorder interface
http://developer.android.com/reference/android/media/MediaRecorder.html

BIP by example: Mutual exclusion

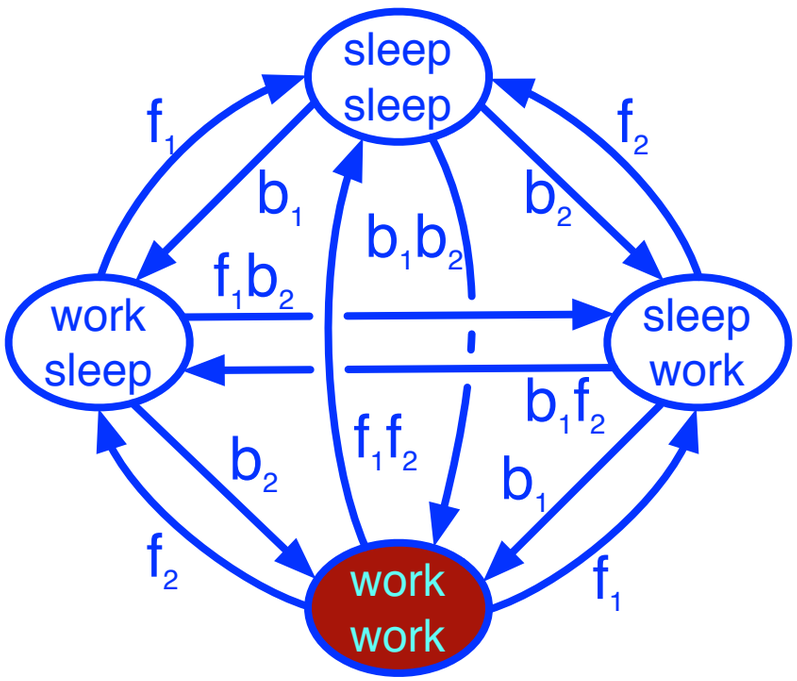


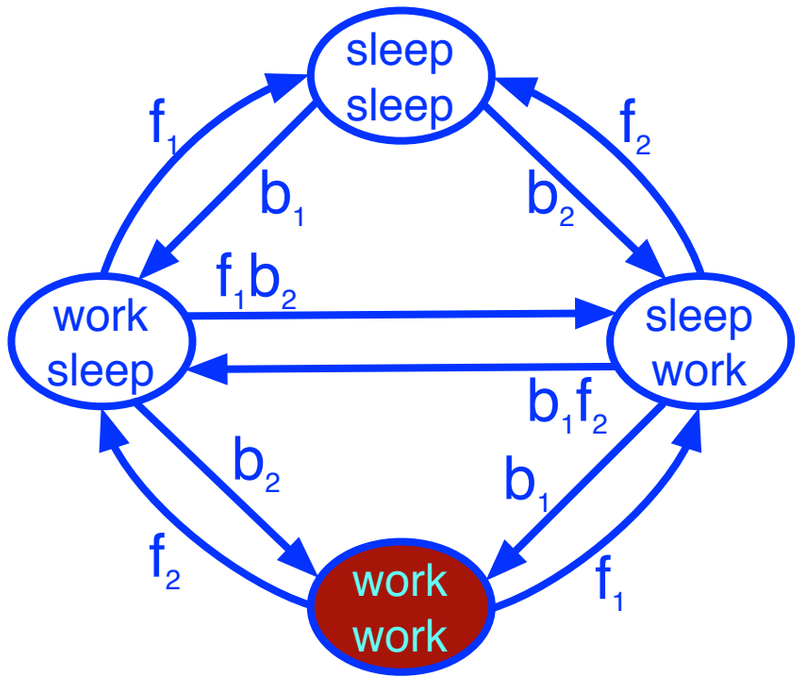
Interaction model:

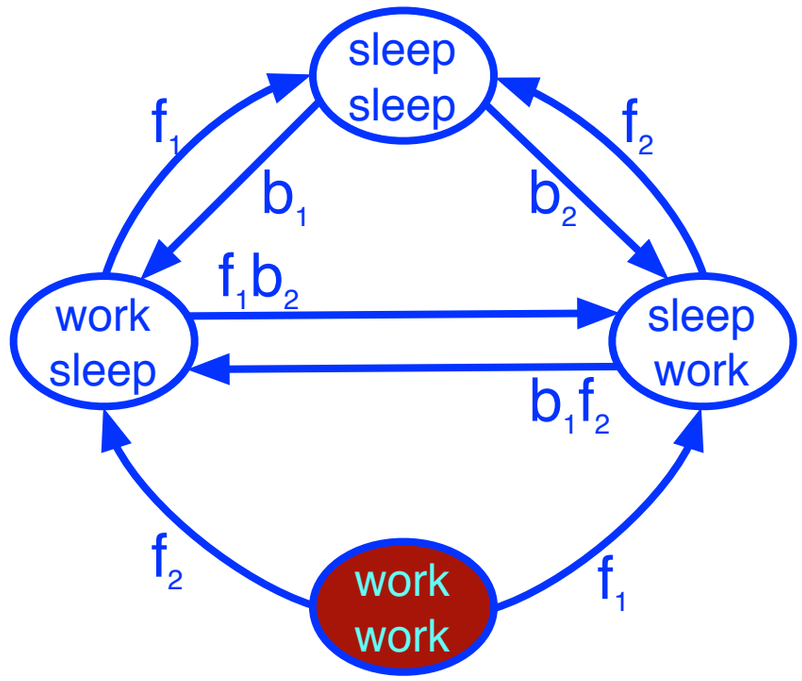
$\{b_1,\ b_2,\ b_1\ f_2,\ b_2\ f_1,\ f_1,\ f_2\}$

Maximal progress:

$b_1 < b_1\ f_2,\ b_2 < b_2\ f_1$







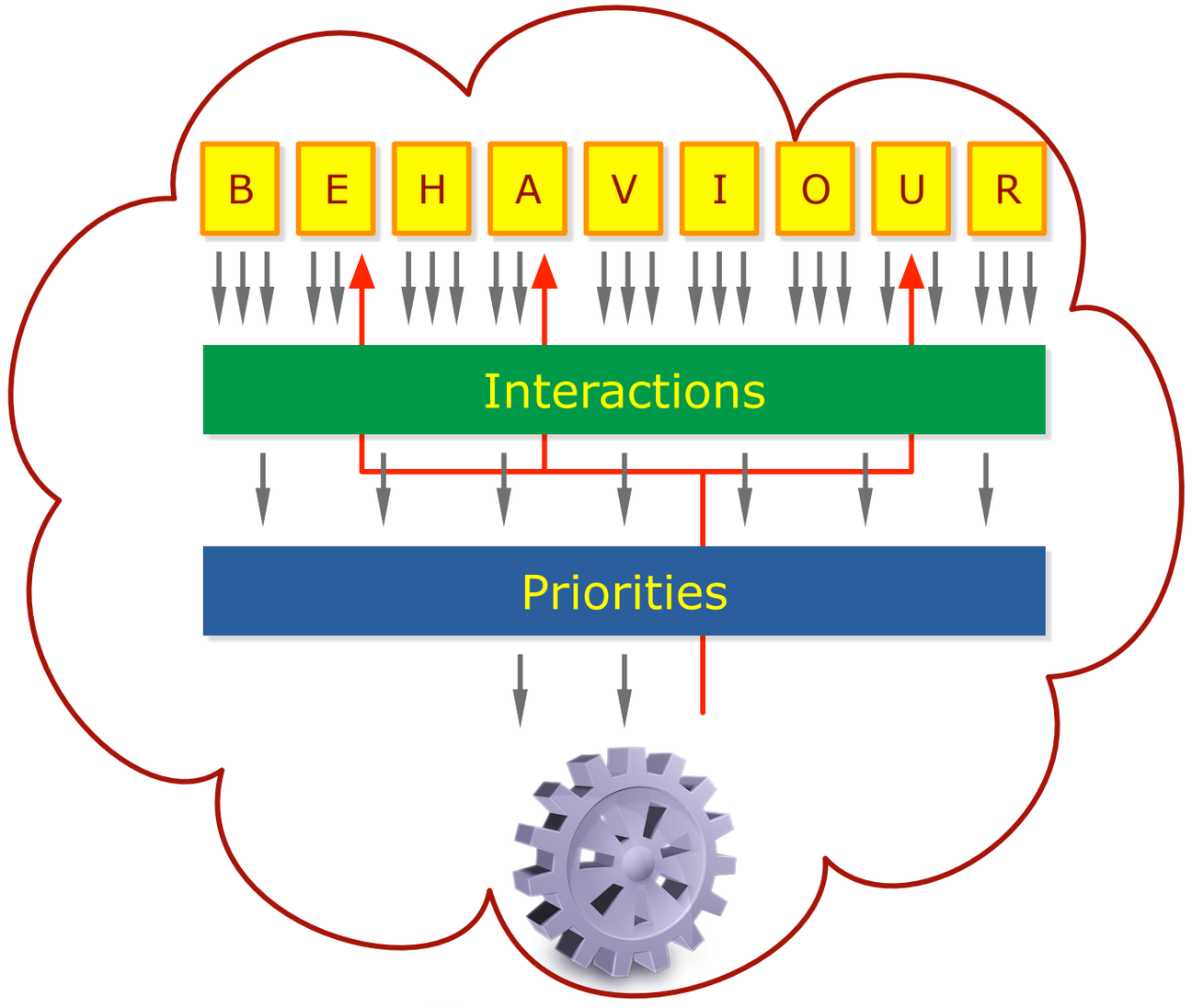
Engine-based execution

1. Atoms reach a stable state.

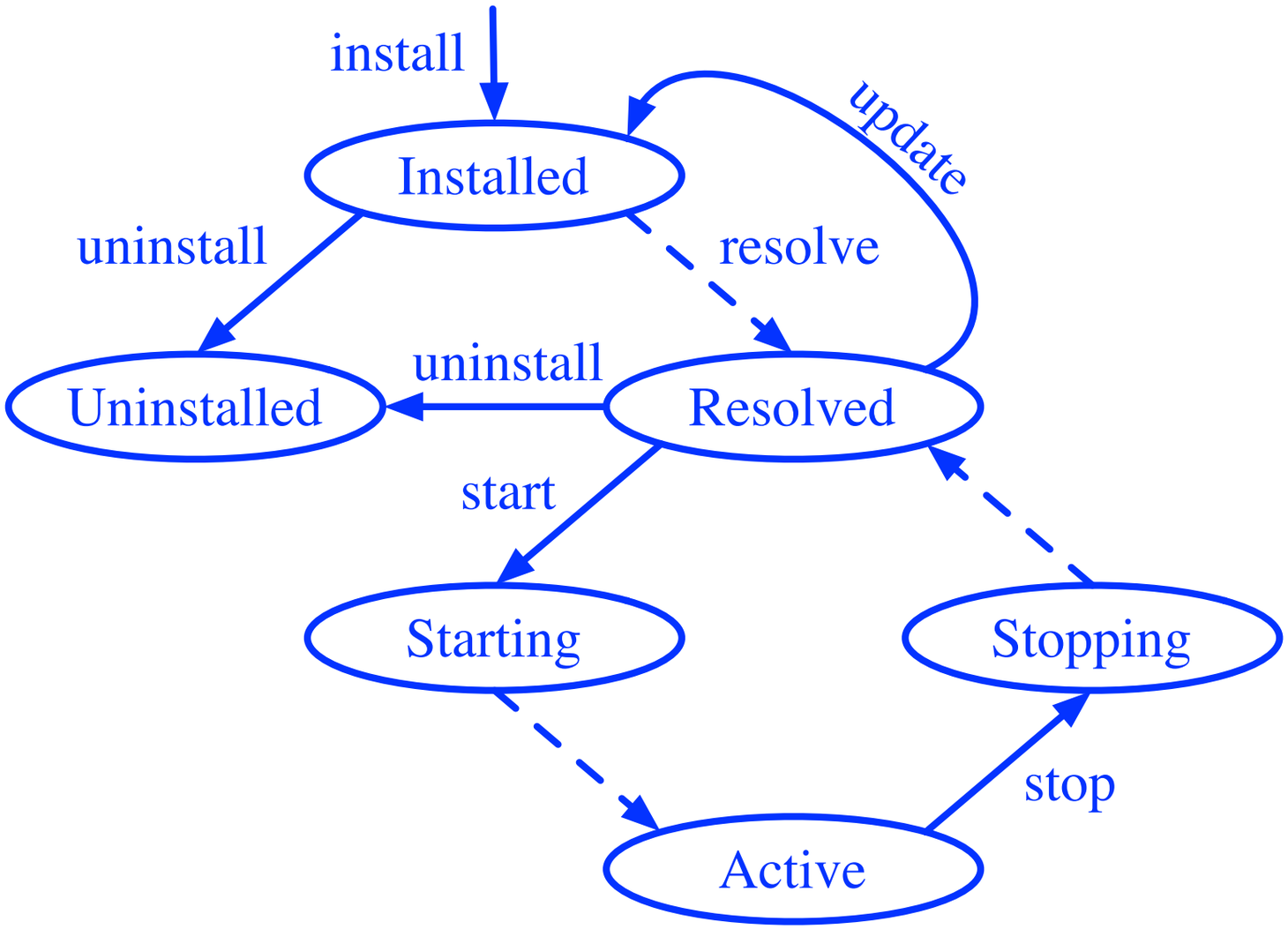
2. Atoms notify the Engine about enabled transitions.

3. The Engine picks one interaction.

4. The Engine notifies the involved components.



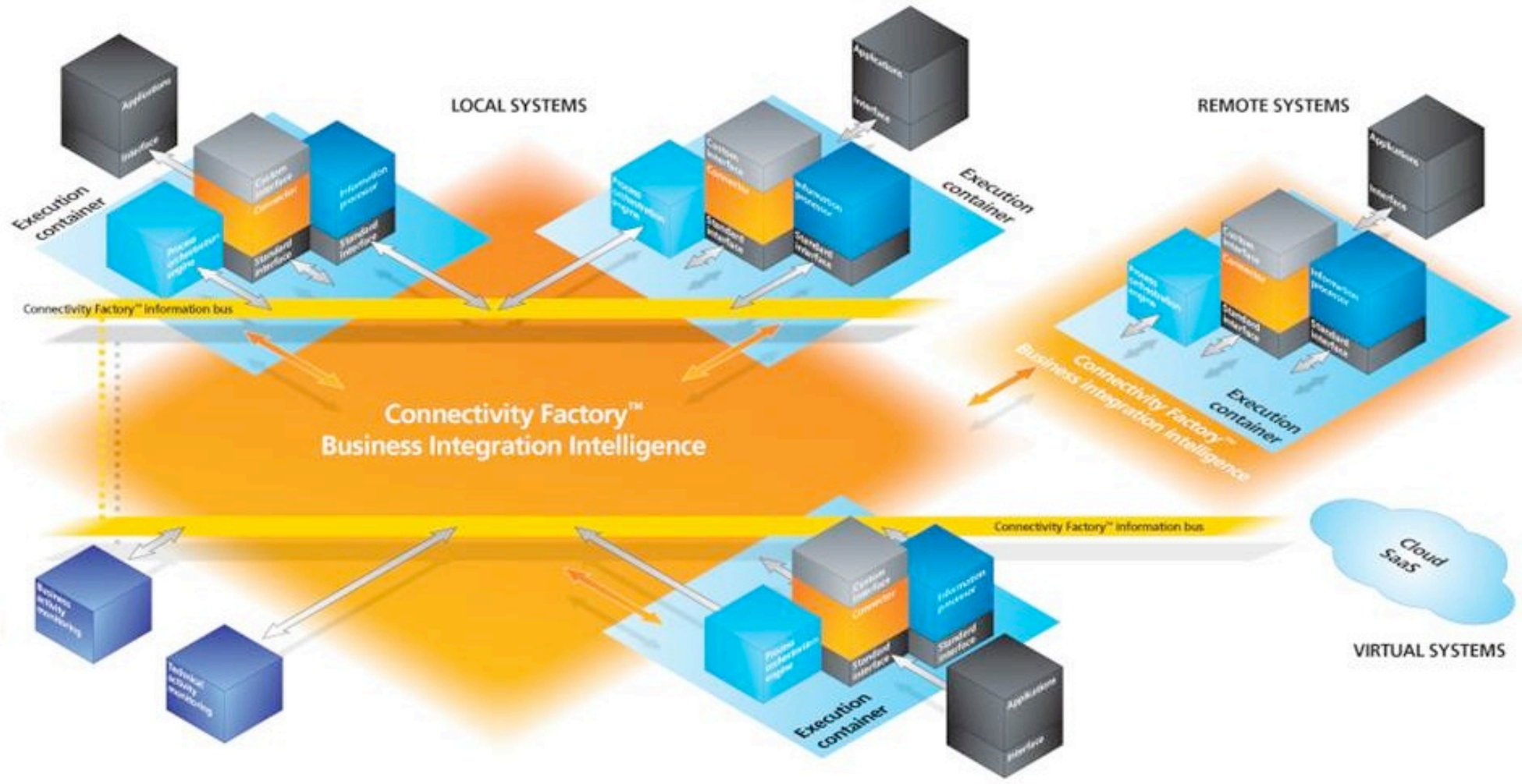
OSGi bundle states



Only lifecycle state is shown.

All functional states are hidden in the ‘Active’ state.

Use case: Camel Routes



Many independent routes share memory

- We have to control the memory usage
- e.g., by limiting to only a safe number of routes simultaneously

Camel API: `suspendRoute` and `resumeRoute`

Camel routes

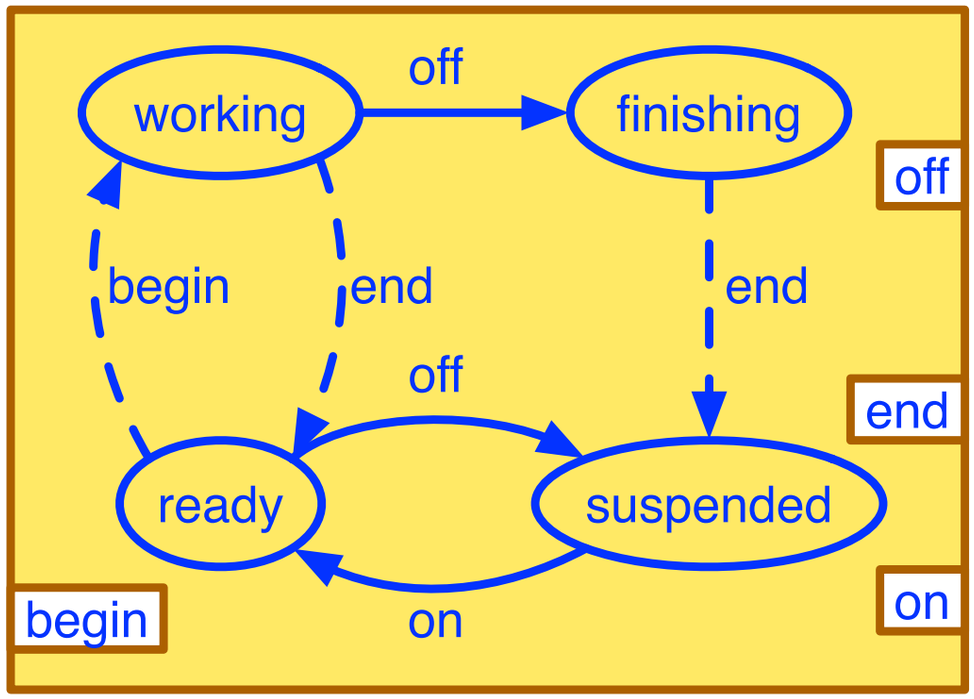
```
public class RouteBuilder(...)
{
  from(…).process(…).to(…);
}
```

Transition types:

- Enforceable

  (can be controlled by the Engine)
- Spontaneous

  (inform about uncontrollable external events)

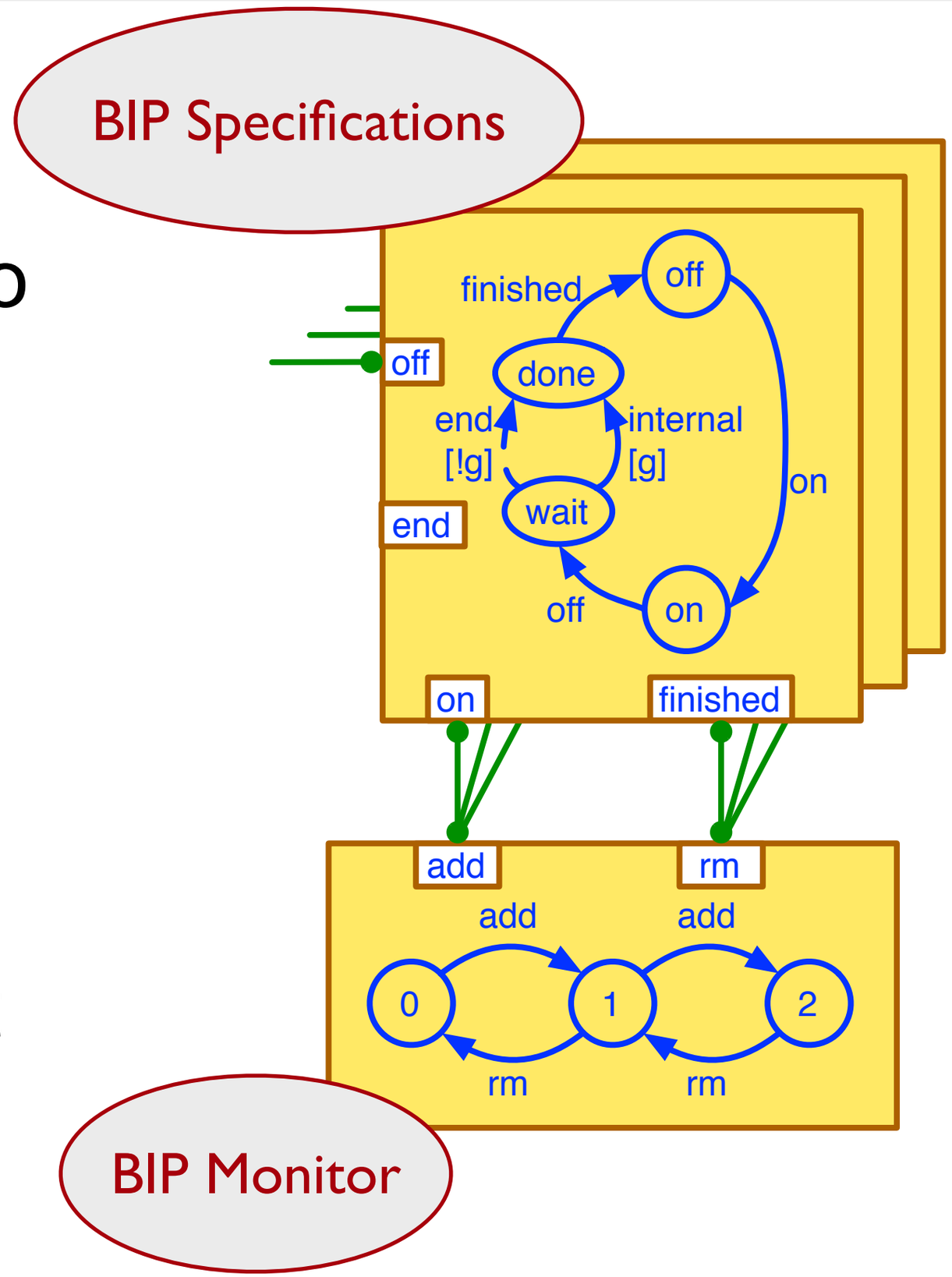


Use case: BIP model

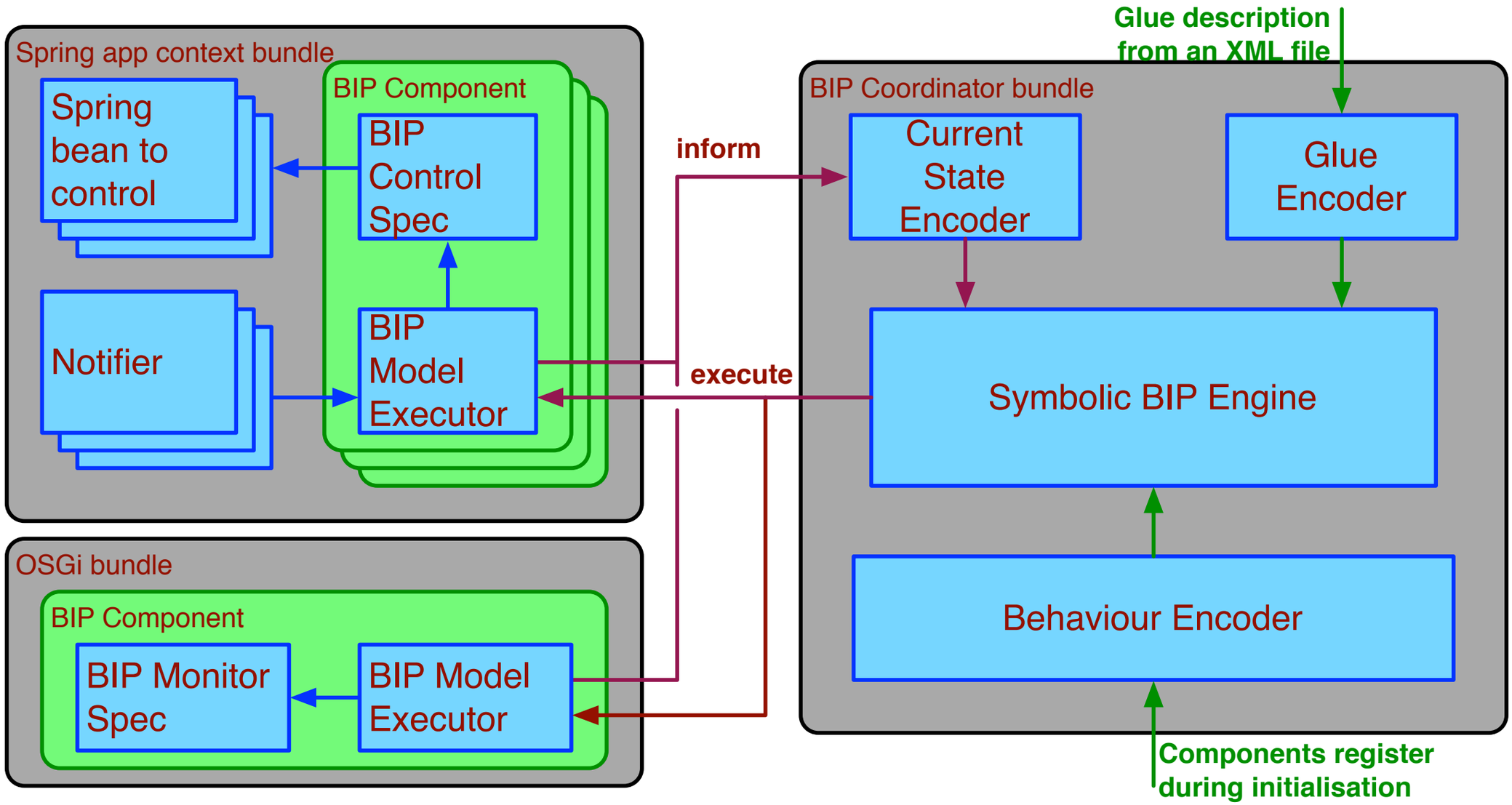
The (enforceable) `off` ports are made visible to the Engine through singleton connectors

The `end` ports correspond to spontaneous events

The Monitor component limits the number of active routes to two



Implemented architecture



Arrows:

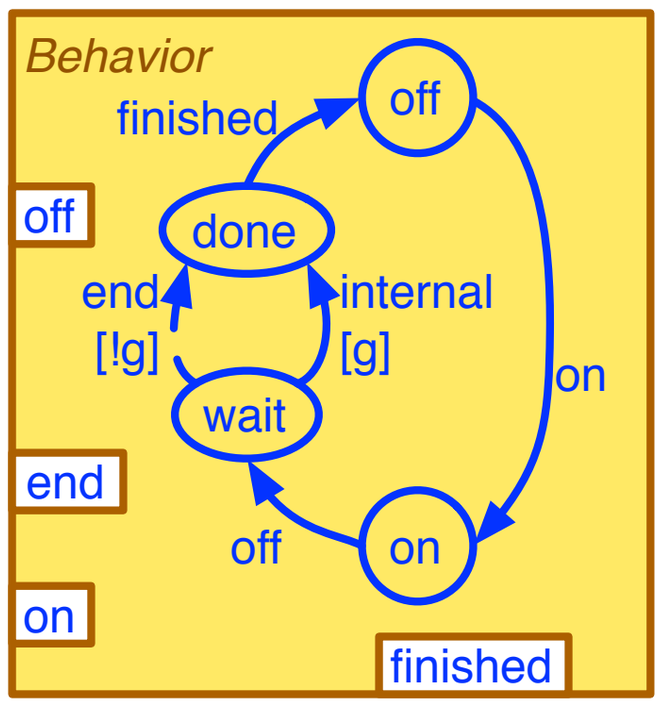
- Blue — API calls between model and entity
- Red — OSGi-managed through published services
- Green — called once at initialisation phase

BIP Component specification: Ports, Initial

```
@bipPorts({
  @bipPort(name = "end", type = "spontaneous"),
  @bipPort(name = "off", type = "enforceable"),
  …
})

@bipComponentType(
  initial = "off",
  name = "org.bip.spec.switchableRoute")

public class SwitchableRoute
   implements CamelContextAware,
              InitializingBean,
              DisposableBean
{ … }
```



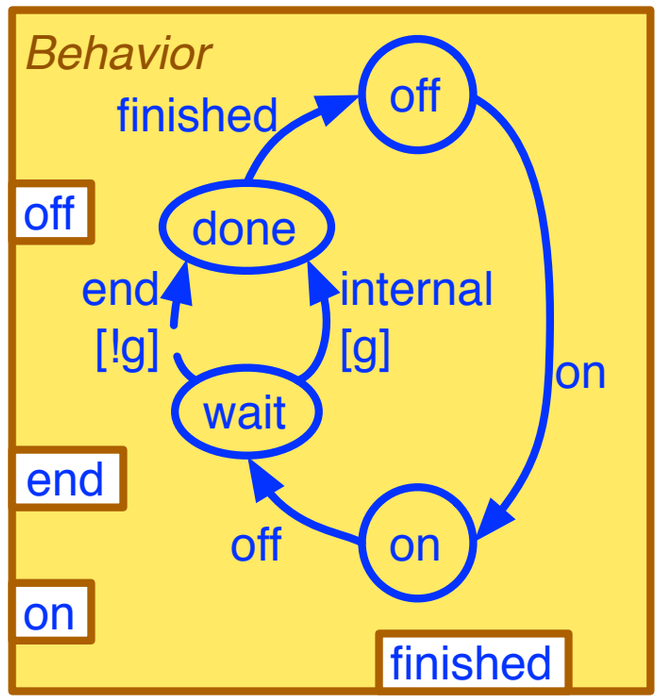
BIP Component specification: Transitions

```
@bipTransition(name = "off",
  source = "on", target = "wait", guard = "")

public void stopRoute() throws Exception {
   camelContext.suspendRoute(routeId);
}
```

Transition annotations provide

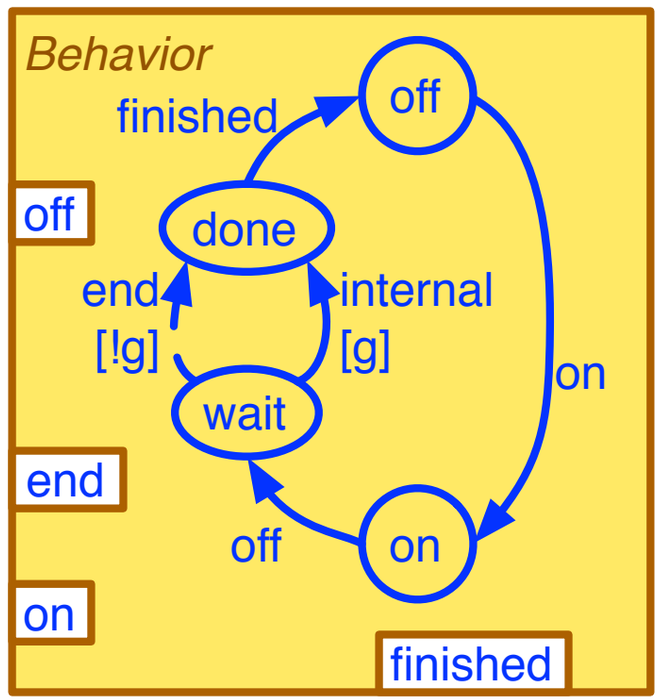
- Label: a port, declared by `@bipPort`
- Source and target states
- Guard expression



BIP Component specification: Guards

```
@bipTransition(name = "end",
   source = "wait", target = "done",
   guard = "!isFinished")
public void spontaneousEnd() throws Exception { … }

@bipTransition(name = "",
   source = "wait", target = "done",
   guard = "isFinished")
public void internalEnd() throws Exception { … }
```

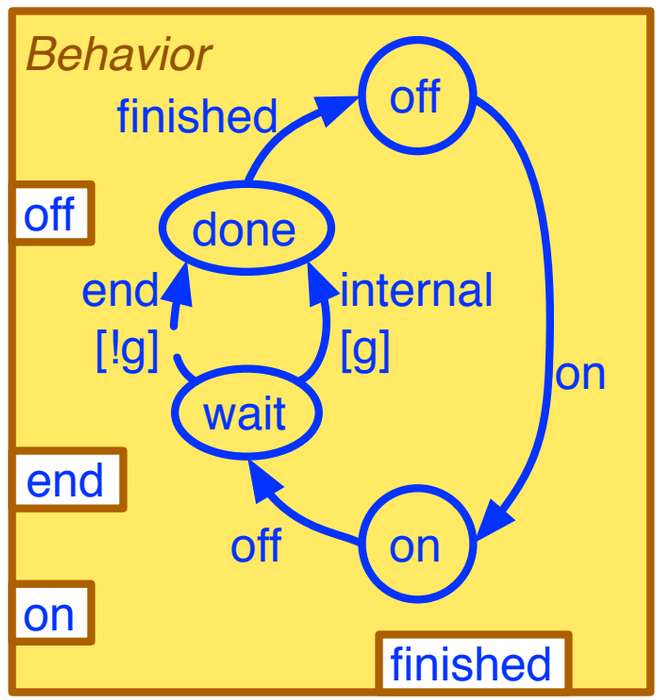


BIP Component specification: Guards

```
@bipTransition(name = "end",
   source = "wait", target = "done",
   guard = "!isFinished")
public void spontaneousEnd() throws Exception { … }

@bipTransition(name = "",
   source = "wait", target = "done",
   guard = "isFinished")
public void internalEnd() throws Exception { … }

@bipGuard(name = "isFinished")
public boolean isFinished() {
   CamelContext cc = camelContext;
   return
      cc.getInflightRepository().size(
         cc.getRoute(routeId).getEndpoint()
      ) == 0;
}
```

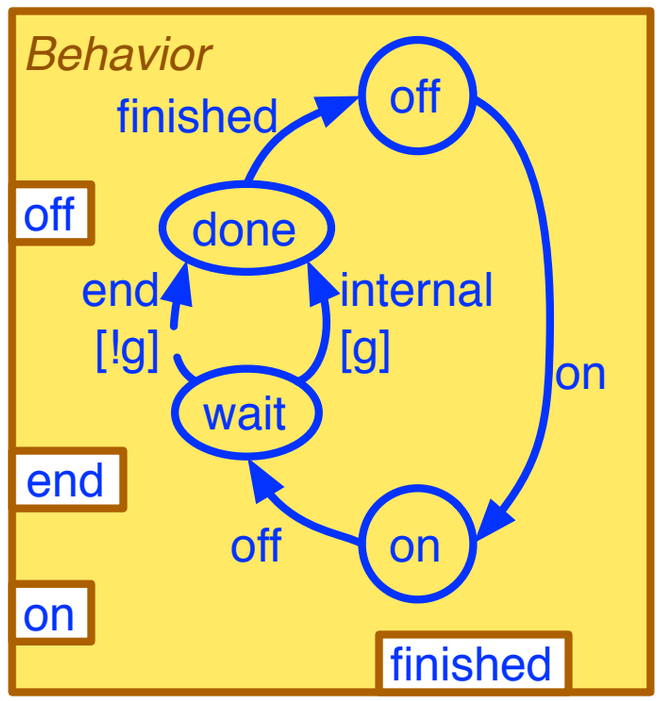


BIP Component specification: Interface

```
public interface BIPComponent extends BIPSpecification
{
   void execute(String portID);
   void inform(String portID);
}
```

Interface methods:

- `execute` — called by the Engine to execute an enforceable transition
- `inform` — called by Notifiers to inform about spontaneous events



BIP Executor: Interface

```
public interface Executor extends BIPComponent {
   void publish();
   void unpublish();

   void register(BIPEngine bipEngine);
   void deregister();
}
```

Interface methods:

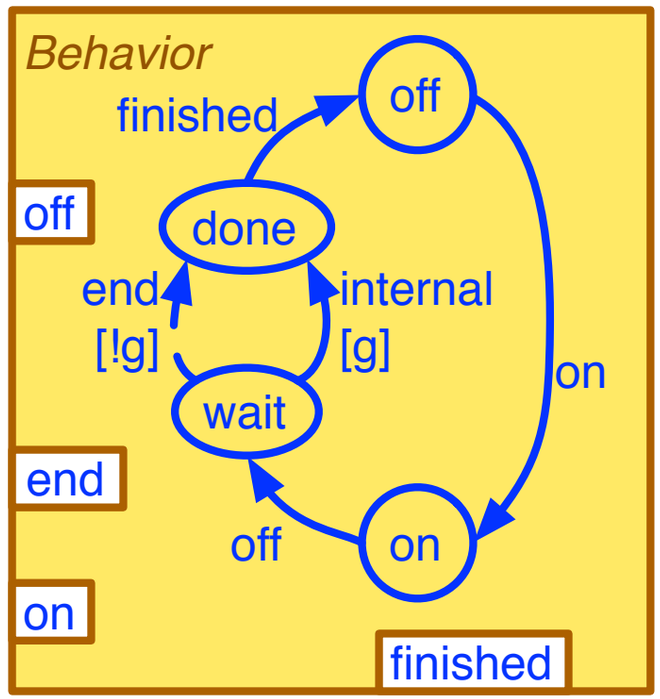
- `publish/unpublish` — collaborates with OSGi service registry
- `register/deregister` — manage connection with the BIP Engine

Implements the component execution semantics

Spontaneous event notifiers

```
new RoutePolicy() {
  …
  public void onExchangeDone(
    Route route, Exchange exchange)
  {
    executor.inform("end");
  }
}
```

BIP spec may require knowledge about it’s executor to set up notification mechanisms



Conclusion (1/2)

- Business components do not have incorporated fragile coordination code that depends on the execution environment.

- Such code is confined to
  - BIP Glue specification
  - BIP Specification of the monitors imposing safety properties

Conclusion (2/2)

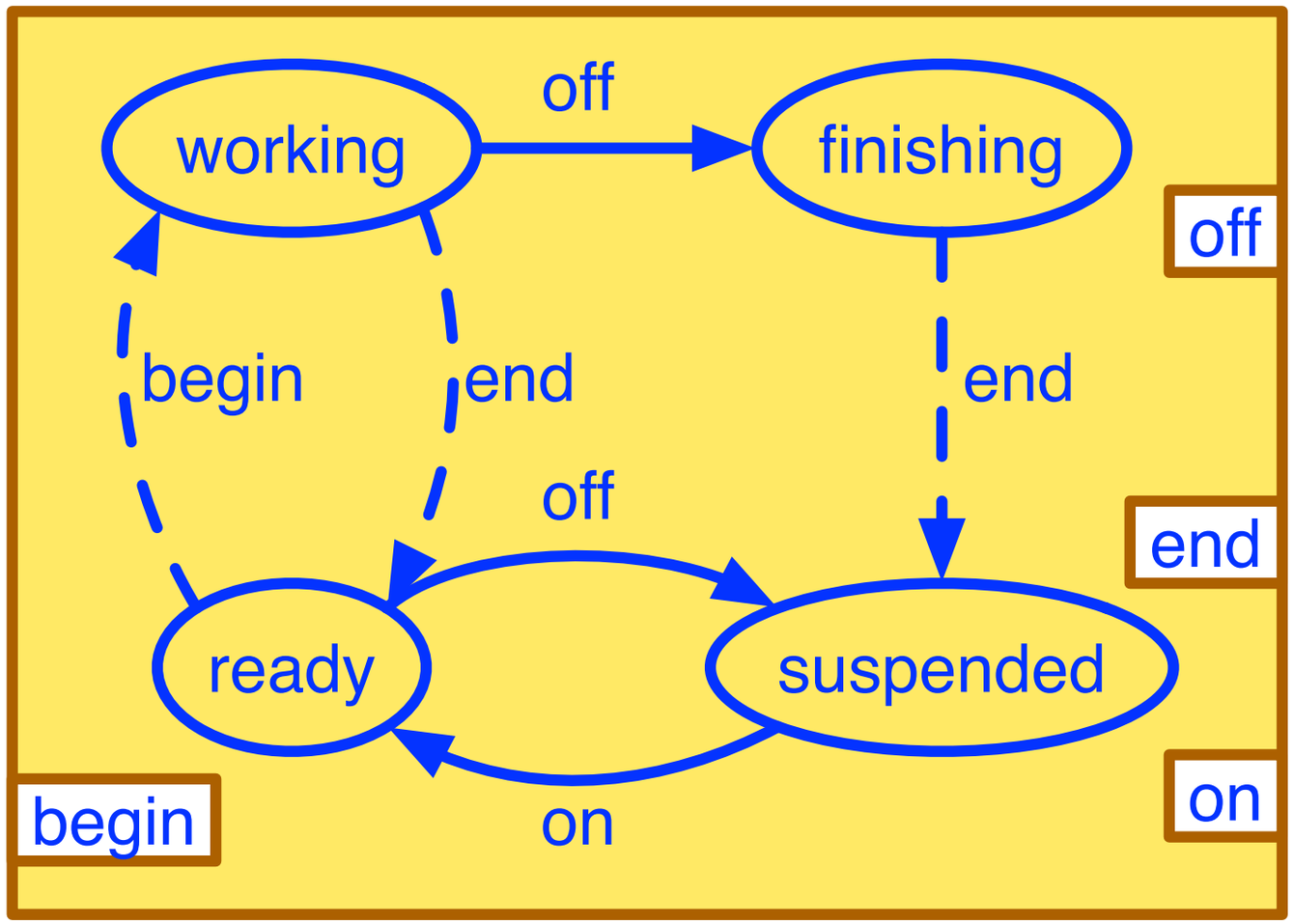
- BIP Specification classes provide reusable specification of the underlying Finite State Machine of the components.

- Component coordination ensuring safe execution of the system is specified as a combination of
  - BIP Specification for the safety properties monitors
  - Allowed interactions between components

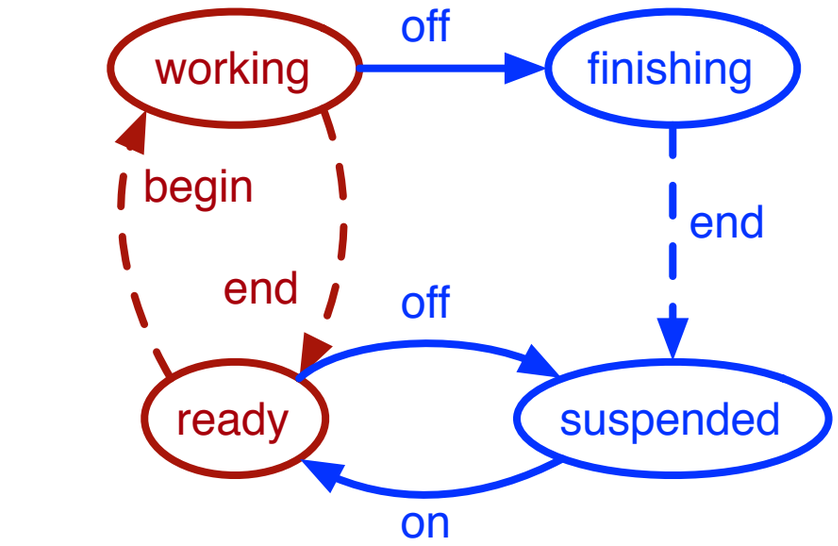
Future work

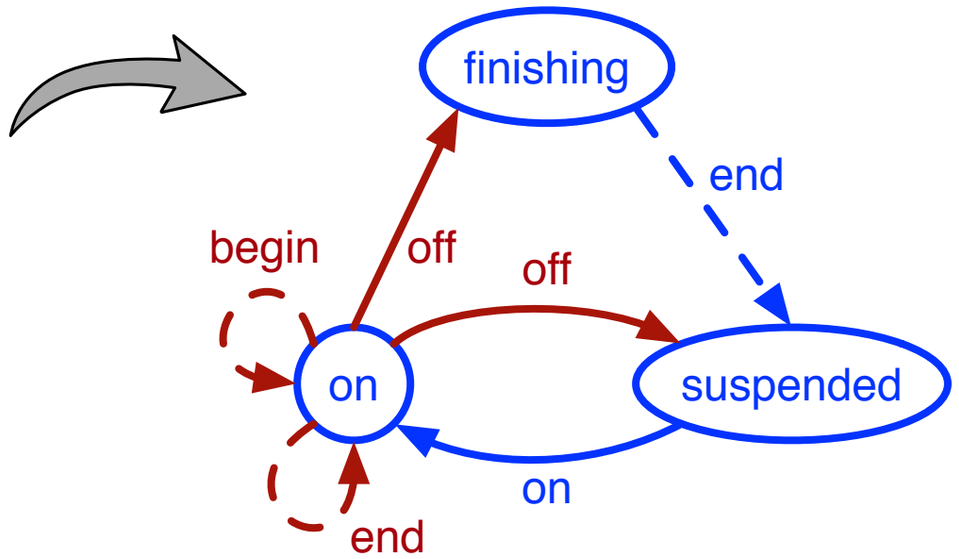
- Data transfer
- Exception handling & transaction support
- Further experimentation with real-life applications
- Adding BIP coordination to the OSGi standard

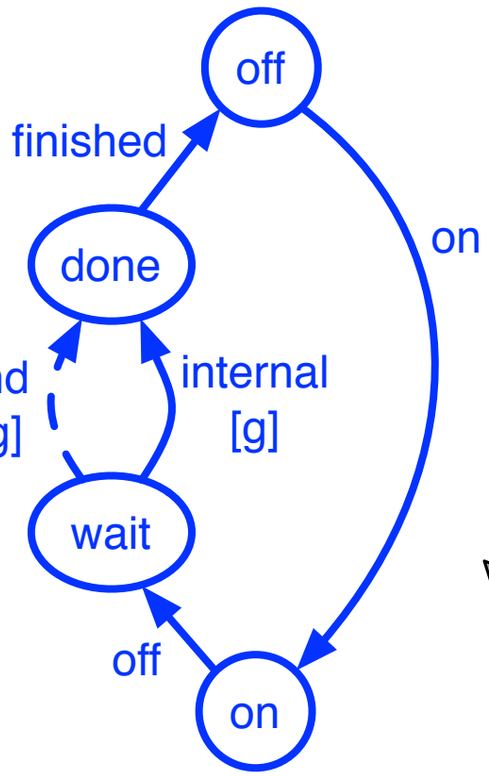
State stability

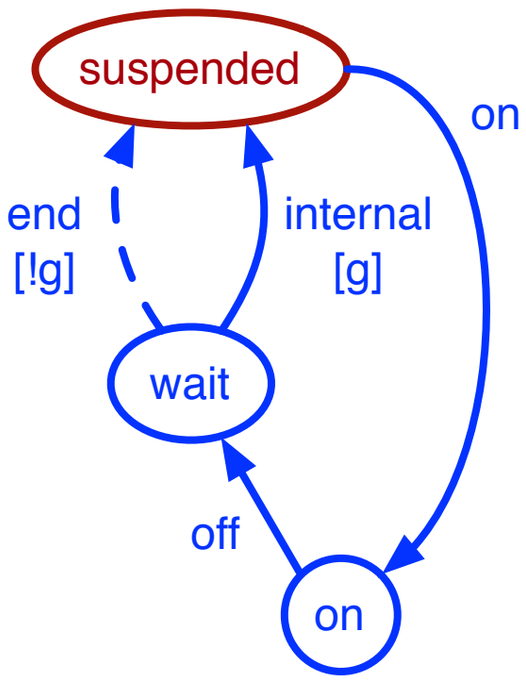


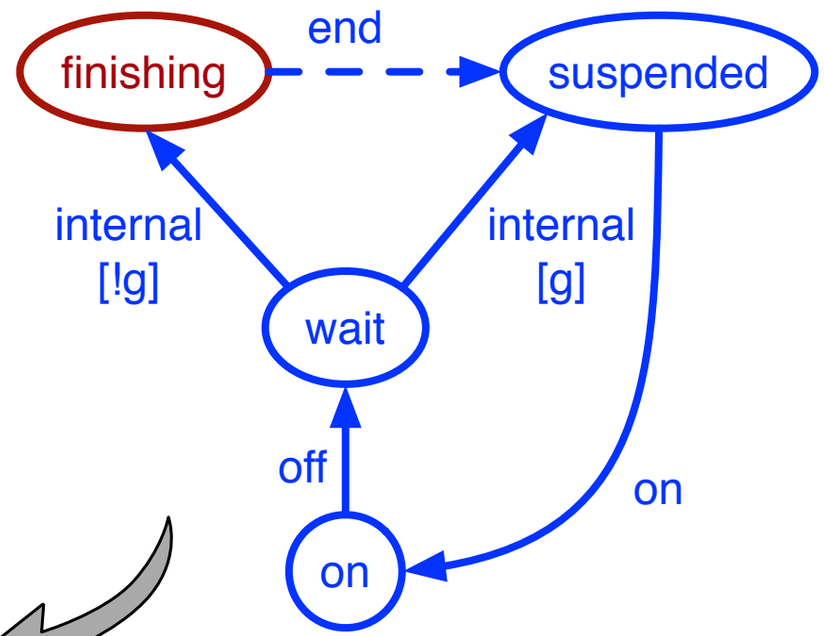
It must be possible to postpone the treatment of ***spontaneous*** events.











g = isFinished()