Planware II

Marcel Becker
Limei Gilham
Douglas R. Smith

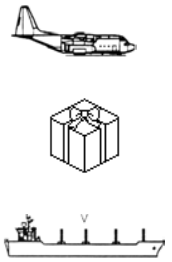
Kestrel Technology
Palo Alto, California

www.kestreltechnology.com

Planware: Synthesis of High Performance Schedulers

Model Construction Tool

Library of resource and task models



Model of Scheduling Problem (mode machine)

250 lines, 4 machines

Planware Scheduler Generator

Customized Scheduler (in MetaSlang)

6560 loc, 632 defs

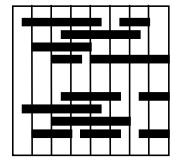
Optimization and Code Generation

Customized Scheduler (in CommonLisp)

19088 loc, 1784 defs

TPFDD data, Resource data

Customized Scheduler



schedule

Kestrel Technology

How can we model resources?

What does a cargo aircraft do? e.g.

prep →fly →offload →prep →fly →fly →offload →fly →park →…

A *behavior* of a resource is a sequence of activities

Each *activity* is represented by a collection of attributes and their values

e.g. A "fly" activity of a cargo aircraft:

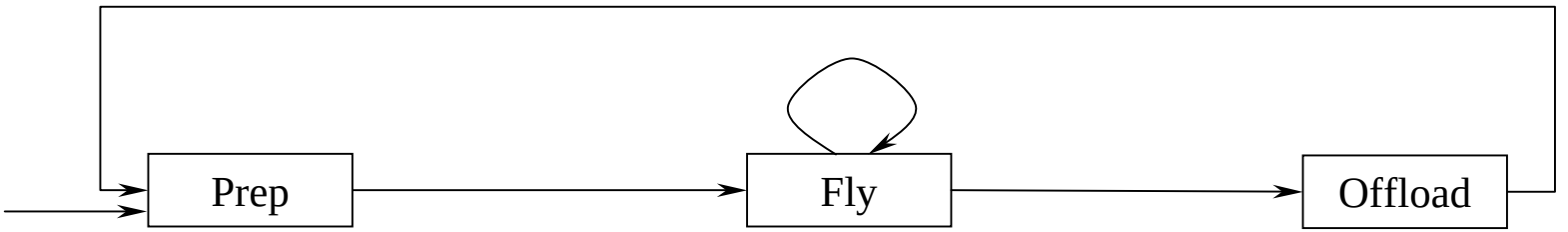
fly	
start-time :	2100 C0
finish-time :	0515 C1
duration :	8hr+15min
origin:	McGuire
destination:	Mildenhall
crew :	Jones
manifest:	{ … }

A *resource* is characterized by its set of behaviors

e.g. cargo aircraft = { …
prep →fly →offload →prep …,
prep →fly →fly →offload →fly →park →… ,
… }

Key Idea:
model resources as state machines

The (possibly infinite set of) behaviors of a resource can be expressed finitely by a state machine!



The *modes* of the state machine model the activities of the resource;
The *transitions* model the restrictions on the sequence of activities

e.g. prep →fly →offload →prep →fly →fly →offload →prep →…
or prep →fly →fly →offload →prep →fly →fly →offload →prep →…

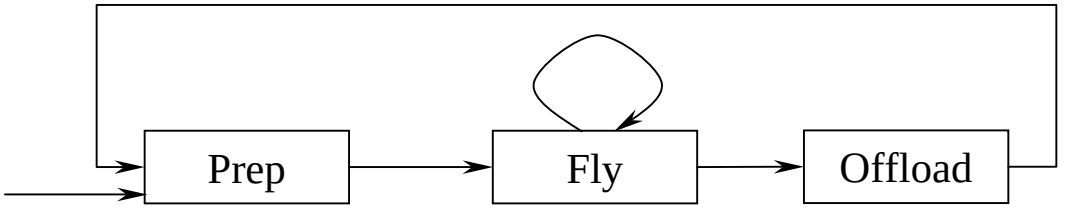
Modeling resources as state machines: *mode variables*

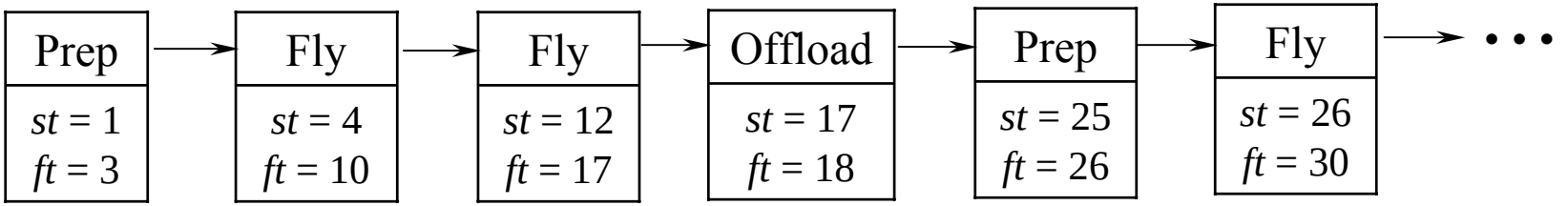
A signature presents the variables relevant to a resource:

Signature for Cargo Aircraft	
var *st* : *Time*	***– start time***
var *ft* : *Time*	***– finish time***
var *dur* : *Duration*	***– duration***
var *orig* : *Port*	***– origin***
var *dest* : *Port*	***– destination***
var *mani* : *set(Task)*	***– manifest***
var *crew* : *Crew*	***– flight crew***
var *fuel-amt* : *Pounds*	***– fuel onload***
const *max-cap* : *Ston*	***– max acl***
var *loc* : *Location*	***– current position***

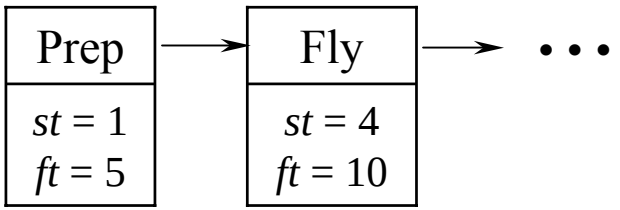
Example Behaviors

Signature for Cargo Aircraft	
var *st* : *Time*	***– start time***
var *ft* : *Time*	***– finish time***
…	



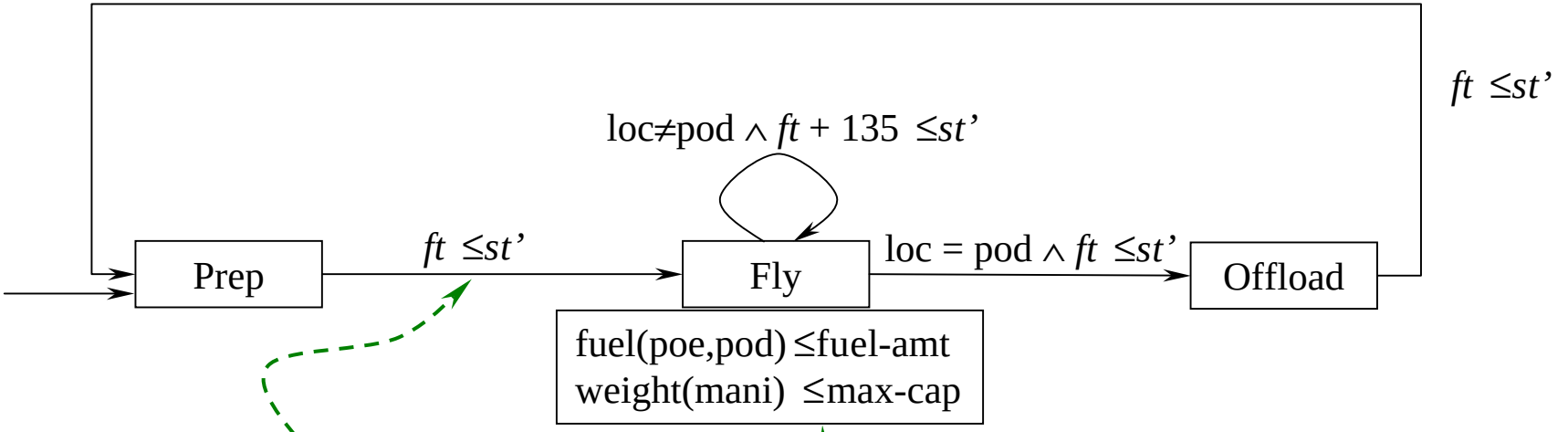


what about this?



Modeling resources as state machines: *constraints*

Constraints on the modes and transitions constrain the evolution of the mode variables



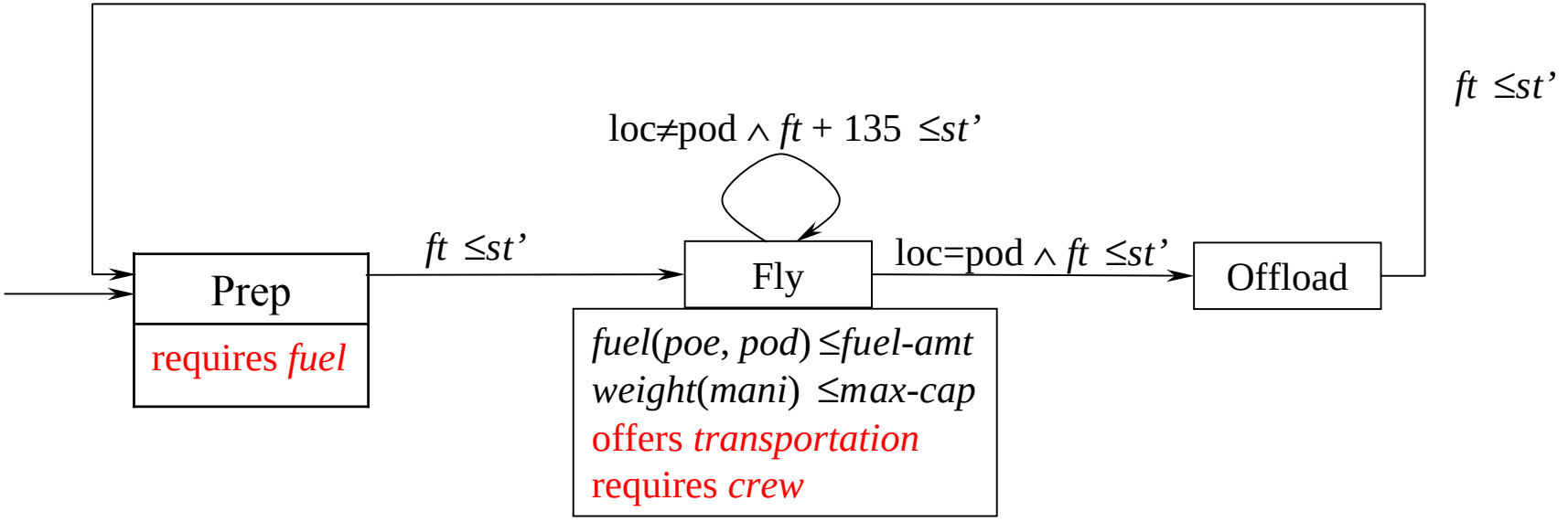
The start time of the new activity is
no earlier than the finish time of the previous activity

When flying, the resource must have adequate fuel
and the weight of the manifest must not exceed capacity

cf. in KIDS: ∀(ac:Aircraft, flt: Flight)
(ac ∈ sched ∧ flt ∈ ac.flights ⇒ weight(flt.mani) ≤max-cap)

Modeling resources as state machines: *services*

Each mode may *offer* services (to accomplish a certain kind of task)
and each mode may *require* services of other resources

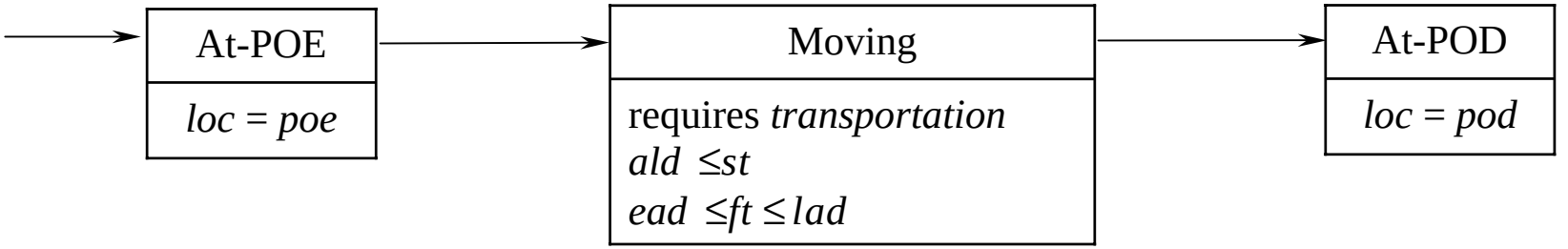


Key Idea: *also model tasks as state machines!*

Effectively, a task is a resource that requires the services of other resources

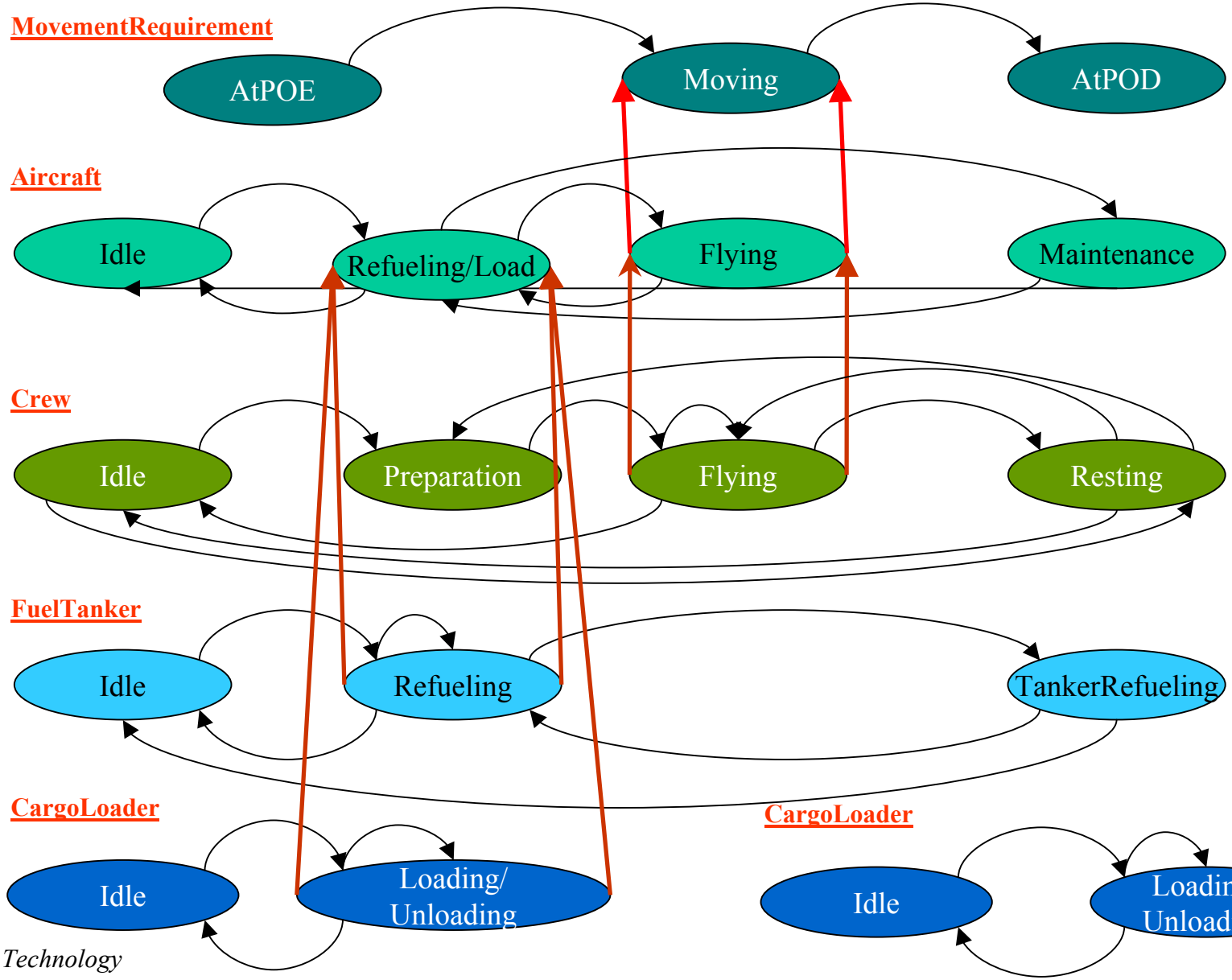
e.g. a TPFDD-like movement requirement:

Signature for Move Req't.
var *st, ft* : *Time* var *dur* : *Duration* const *poe, pod* : *Port* const *ald, ead, lad* : *Time* const *demand* : *Capacity* var *loc : Location*

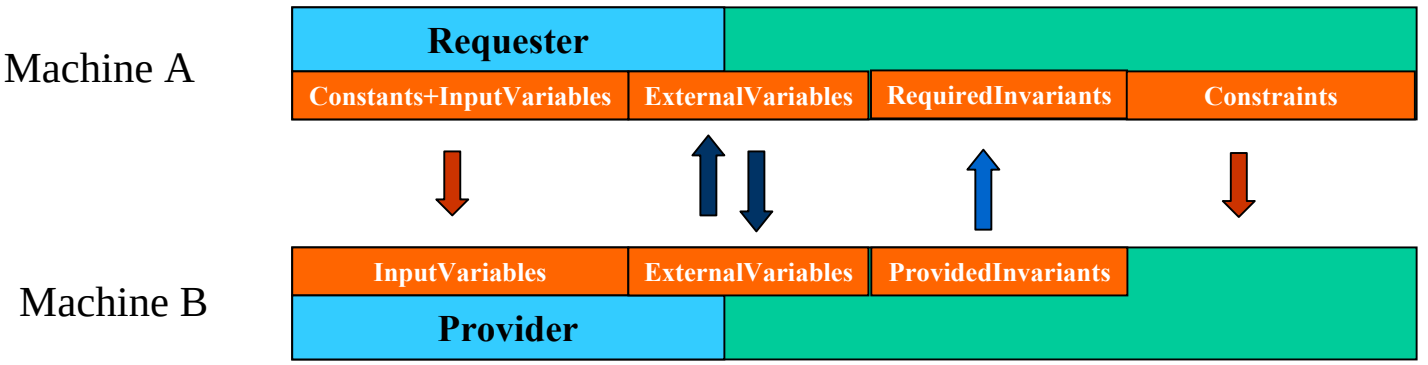


The scheduling process is motivated by the need to drive each task machine to a final/completed state, which entails scheduling the required resources, etc.

Hierarchy of Required Services



Matching of Services



$$\forall(\text{constants(A)}, \text{input-vars(A)}, \text{constants(B)})$$
$$\exists(\text{external-vars(A)}, \text{internal-vars(A)},$$
$$\text{input-vars(B)}, \text{ext-vars(B)}, \text{internal-vars(B)})$$
$$(\text{ProvidedConditions(A)}, \text{ProvidedConditions(B)})$$
$$\Rightarrow$$
$$(\text{RequiredConditions(A)}, \text{Constraints(A)},$$
$$\text{RequiredConditions(B)}, \text{Constraints(B)})$$

MovementRequirement + Aircraft

∀(constants(A), input-vars(A), constants(B)) ∃ (ext-vars(A), internal-vars(A), input-vars(B), ext-vars(B), internal-vars(B))
(ProvidedConditions(A), ProvidedConditions(B))
⇒
(RequiredConditions(A), Constraints(A), RequiredConditions(B), Constraints(B))

∀(**theMvr, POE, POD, ALD, LAD, EAD, theAc, maxFuel, maxCargoCap**)
∃(**ac, st_{mvr} ,ft_{mvr}, dur_{mvr}, mvr, initialAcLocation, finalAcLocation, fuelLevel, st_{ac}, ft_{ac}, dur_{ac}, accFlyingHours, timeSinceMaint, currLocation, manifest, crew**)

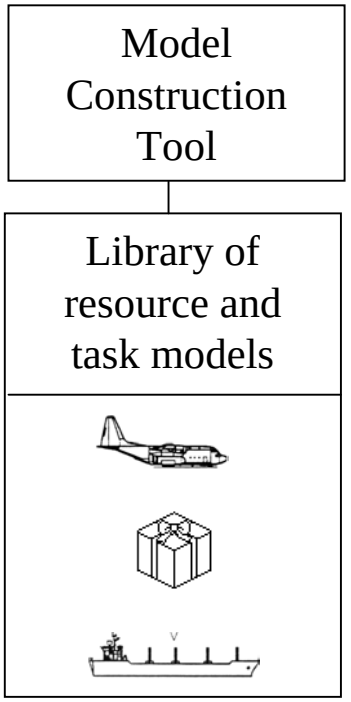
POE != POD,
CargoAtLocation(theMvr, POE, st_{mvr},st_{mvr}), CargoAtLocation(theMvr, POD,ft_{mvr}, ft_{mvr}),
inTransit(mvr, theAc, initialAcLocation, finalAcLocation, st_{ac},ft_{ac}),
inManifest(mvr, manifest),

⇒

inTransit(theMvr, ac, POE, POD, st_{mvr}, ft_{mvr})
st_{mvr} >= ALD, ft_{mvr} >= EAD, ft_{mvr} <= LAD,
initialAcLocation != finalAcLocation, atLocation(theAc, initialAcLocation, st_{ac}, st_{ac}),
hasFuel(theAc, fuelLevel, st_{ac}, st_{ac}) , insideTransporter(mvr, theAc,st_{ac},ft_{ac}), assignedCrew(theAc, crew, st_{ac},ft_{ac})
timeDuration(ft_{ac},st_{ac}) >= transportDuration(theAc, initialAcLocation, finalAcLocation),
fuelLevel > requiredFuel(theAc, initialAcLocation, finalAcLocation)

mvr := theMvr, ac := theAc, initialAcLocation := POE, finalAcLocation := POD, st_{mvr} := st_{ac}, ft_{mvr} := ft_{ac} ,
st_{ac} >= ALD, ft_{ac} >= EAD, ft_{ac} <= LAD,
atLocation(theAc, POE, st_{ac}, st_{ac}),
hasFuel(theAc, fuelLevel, st_{ac}, st_{ac}), insideTransporter(theMvr, theAc,st_{ac},ft_{ac}), assignedCrew(theAc, crew, st_{ac},ft_{ac})
timeDuration(ft_{ac},st_{ac}) >= transportDuration(theAc, initialLocation, finalLocation),
fuelLevel > requiredFuel(theAc, initialLocation, finalLocation), maxFuel >= fuelLevel

Code Generation Process



Model of Scheduling Problem
(mode machine)

250 lines, 4 machines

Planware Scheduler Generator

Customized Scheduler
(in MetaSlang)

6560 loc, 632 defs

Optimization and Code Generation

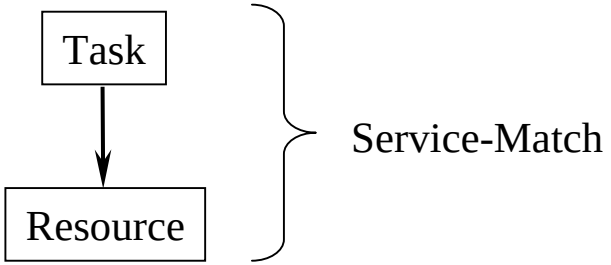
Customized Scheduler
(in CommonLisp)

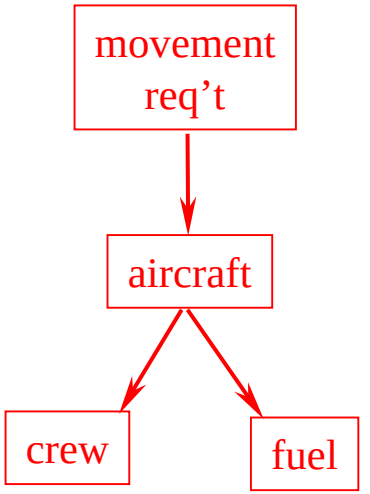
19088 loc, 1784 defs

Code Generation Statistics

Problem	Model LOC	MetaSlang LOC	CommonLisp LOC
Mvr, AC	161	3767	13,833
Mvr, AC, Crew	295	6167	20,225
Mvr, AC, Crew, Parking	444	8734	27,469
Mvr, AC missions, Crew, Port load/unload	536	12377	40,934

Simplified Algorithm Schema





to schedule a task wrt a Service-Match:

get bids from all Resource instances;

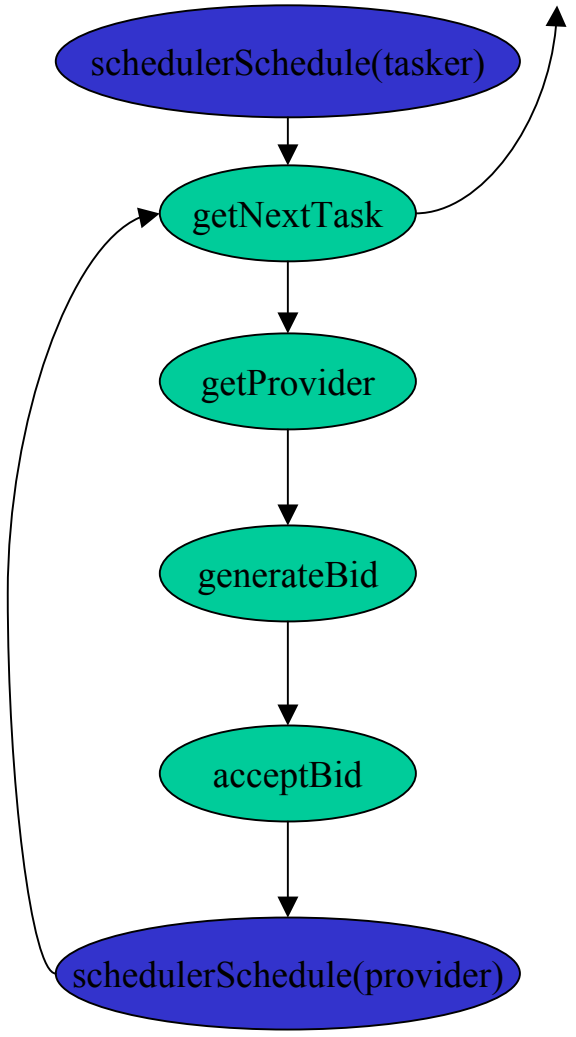
consider the bids in sorted order

if the bid is feasible (via propagation)
and the bid's resource generates a subtask

then schedule the subtask wrt its Service-Match

until one bid succeeds

Program Schema I

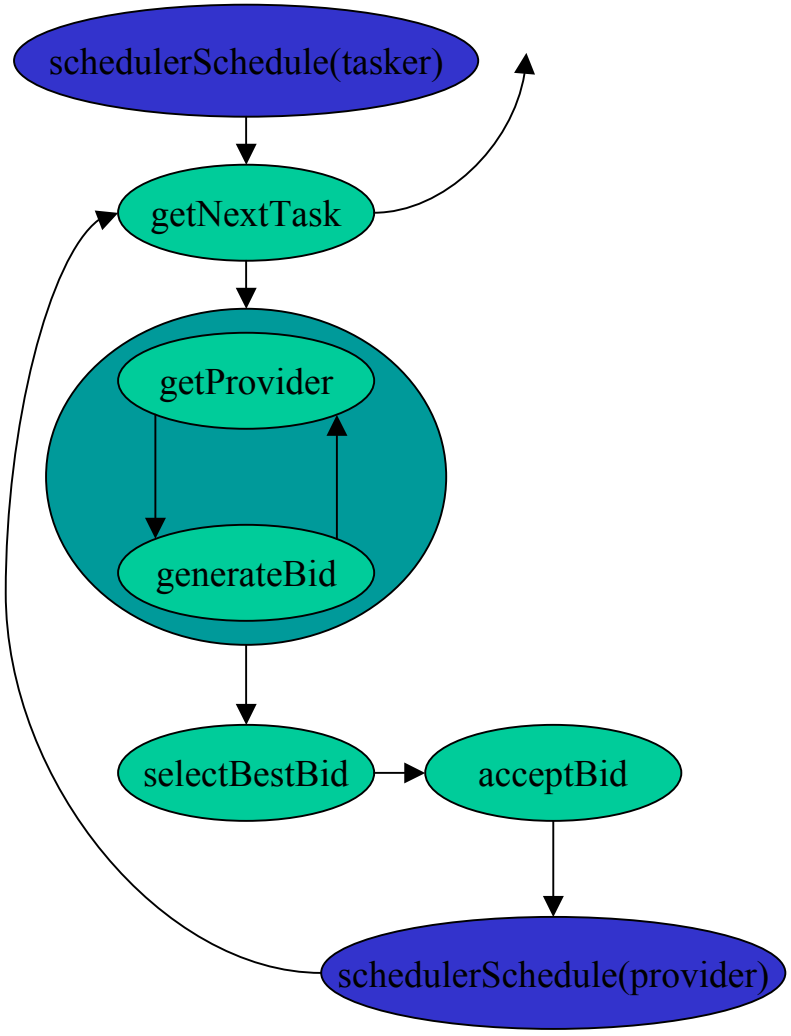


Program Schema:

```
spec SchedulerSchedule(tasker: Tasker) =
  while (nextTask := tasker.getNextTask())
    do {
       while (nextTask.unscheduled())
       do {
          provider := getNextProvider(nextTask)
          if nextTask.satisfyRequirements(provider)
          then {
             bid := provider.generateBid(nextTask)

               if feasible(bid)
               then {
                     tasker.acceptBid(bid)
                     provider.AcceptBid(bid)
                     schedulerSchedule(provider)
                     }
       }
end-spec
```

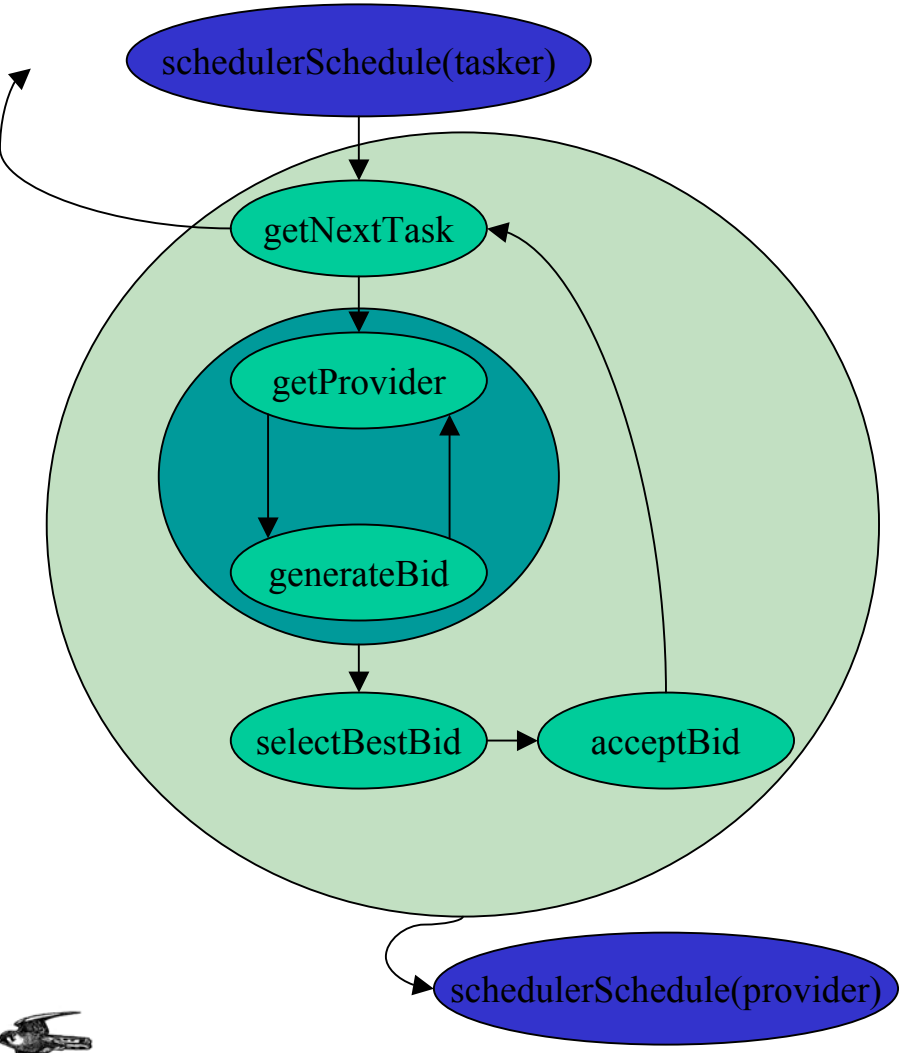
Program Schema II



Program Schema:

```
spec SchedulerSchedule(tasker: Tasker) =
  while (nextTask := tasker.getNextTask())
    do {
       providers := findProvidersForTask(nextTask)
       bids := []
       for provider in providers
       do {
          if nextTask.satisfyRequirements(provider)
          then {bids += provider.generateBid(nextTask) }
          }
       bestBid := getBestBid(bids)
       resource := bestBid.getResource()
       if feasible(bestBid)
       then {
             tasker.acceptBid(bestBid)
             resource.AcceptBid(bestBid)
             schedulerSchedule(resource)
             }
       }
end-spec
```

Program Schema III



Program Schema:

```
spec SchedulerSchedule(tasker: Tasker) =
  resources := []
  while (nextTask := tasker.getNextTask())
    do {
        providers := findProvidersForTask(nextTask)
        bids := []
        for provider in providers
        do {
            if nextTask.satisfyRequirements(provider)
            then { bids += provider.generateBid(nextTask)
            }
        bestBid := getBestBid(bids)
        resource := bestBid.getResource()
        if feasible(bestBid)
        then {
              tasker.acceptBid(bestBid)
              resource.AcceptBid(bestBid)
              resources += resource
              }
        }
        for resource in resources
        do { schedulerSchedule(resource) }
end-spec
```

Program Instance – Composing Program Schemas

```
program schedulerSchedule(tasker: VideoProcessor) =

  while (nextTask := tasker.getNextTask())
    do {
        while (nextTask.unscheduled())
        do {
            provider := getNextProvider(nextTask)
            if nextTask.satisfyRequirements(provider)
            then {
                bid := provider.generateBid(nextTask)
                  if feasible(bid)
                  then {
                          tasker.acceptBid(bid)
                          provider.AcceptBid(bid)
                        while (nextTask := provider.getNextTask())
                          do {
                              providers := findProvidersForTask(nextTask)
                              bids := []
                              for provider in providers
                              do {
                                  if nextTask.satisfyRequirements(provider)
                                  then {bids += provider.generateBid(nextTask) }
                                  }
                               bestBid := getBestBid(bids)
                               resource := bestBid.getResource()
                               if feasible(bestBid)
                               then {
                                       tasker.acceptBid(bestBid)
                                       resource.AcceptBid(bestBid)
                                       schedulerSchedule(resource)
                                      }
                              }
                          }
                  }
```

Demonstration
Steps in a doing a new project

- In Explorer: Create a new project folder
- In graphic window: build the machines (modes, transitions) and services
- In Explorer: designate initial and final modes
- In text window: for each machine, add constants, vars, constraints;
  add service parameters if necessary
- In text window: add initialization transitions
  fill out the other transitions
- In Explorer: generate service profile
  generate/compile/execute code

Future Directions

- Modeling
  - asynchronous servicing and subtasking via *multithreaded state machines*!
  - improved integration with user-supplied components
  - modeling objectives, preferences, priorities, and heuristics
- Code Generation
  - improved optimizations, data structures, strategies
  - algorithms for incremental rescheduling, distributed scheduling, online schedulin
  - generate C, Java
- Planware environment

  high quality total environment for modeling, scheduling, analysis, visualization, rescheduling

Extras