PAR Implementation Details

Jan M. Allbeck

11/6/2008

Outline

- Creating Objects (and Agents)
- Creating an Environment
- Creating Actions
- Processing Actions
- Projects and Future Work

Creating Objects

- Model them
- Convert them to obj format
- Find location in 

- Add Parent (instances are automatic)
- Edit parameters
- Enter sites
- Agents—roles, groups

Creating an Environment

- ASCII Environment File
  - Room layout -> Cell and Portal Graph
  - Textures
  - Objects
- ASCII Map File
  - Navigate from any room to any room

ASCII Environment File

- Object name (relates to parent in Actionary)
- Room name (relates to parent in Actionary)
- Obj file name
- Check for collisions? (0 = no, 1 = yes)
- X, Y, Z position
- X, Y, Z orientation
- Scale factor
- Used to be bounding radius (no longer used)

Creating Actions

- Create animations (key framed or procedural)
- uPARs vs. iPARs
- 

- Python code
- C++ code: register MGs

uPARs vs. iPARs

- uPARs are uninstantiated or general PARs
- uPARs are the semantics of the action
- iPARs are an implementation of those semantics (fill in the required blanks)
- iPARs are uPARs that have been instantiated with an agent and objects, etc.
- iPARs are ready to be processed
- iPARs can override parameters of the uPARs they are created from.

Adding a uPAR to the Actionary

- Determine new id and a name
- Add Python file names
- Determine proper placement in the hierarchy (parent id)
- Specify duration?
- Specify a default priority
- Specify the number of object participants
- Specify a site type
- Add capability for the agents

Python Files

- Conditionals
- Termination conditions
- Applicability conditions
- Preparatory specifications
- Execution steps

- Post assertions
- During condition
- Purpose achieve

InspectCul.py

```
def culmination_condition(self, agent, obj):
    return 0
```

InspectApp.py

```
def applicability_condition(self, agent, obj):
    if checkCapability(agent, self.name, [obj]):
        return 1
    else:
        return 0
```

InspectPre.py

```
def preparatory_spec(self, agent, obj1):
    radius = getBoundingRadius(obj1);
    distnce = dist(agent, obj1);
    if(distnce > radius):
        actions = {'PRIMITIVE':("Walk",{'agents':agent, 'objects':obj1})};
        return actions
    else:
        return 1
```

InspectExec.py

```
def execution_steps(self, agent, obj):
    actions = {'PRIMITIVE':("Inspect",{'agents':agent, 'objects':obj})};
    return actions
```

C++ Code: Predicates

```
static PyMethodDef prop_methods[] = {
   {"getIntProperty", prop_getIntProperty, METH_VARARGS},
   {"setIntProperty", prop_setIntProperty, METH_VARARGS},
   {"checkCapability", agent_checkCapability, METH_VARARGS},
   {"contain", object_contain, METH_VARARGS},
   {"findObject", prop_findObject, METH_VARARGS},
   {"getVector", prop_getVector, METH_VARARGS},
   {"setVector", prop_setVector, METH_VARARGS},
   {"getPosture", prop_getPosture, METH_VARARGS},
   {"getStatus", prop_getStatus, METH_VARARGS},
   {"getLocation", prop_getLocation, METH_VARARGS},
   {"dist", prop_dist, METH_VARARGS},  //distance between two objects
   {"getBoundingRadius", prop_getBoundingRadius, METH_VARARGS},
   {"inFront", prop_inFront, METH_VARARGS},  //is obj1 in front of obj2
   {"testAppCond", prop_testAppCond, METH_VARARGS},
   {"testCulCond", prop_testCulCond, METH_VARARGS},
   {"testPreSpec", prop_testPreSpec, METH_VARARGS},
   {"nullCheck", prop_nullCheck, METH_VARARGS},
…
   {NULL, NULL}
};
```

C++ Code: Register MGs

- actionTable.addFunctions("Inspect",
  &DoInspect, &CheckInspect);

- DoInspect = function that starts the execution
- CheckInspect = function that indicates whether or not the action is completed

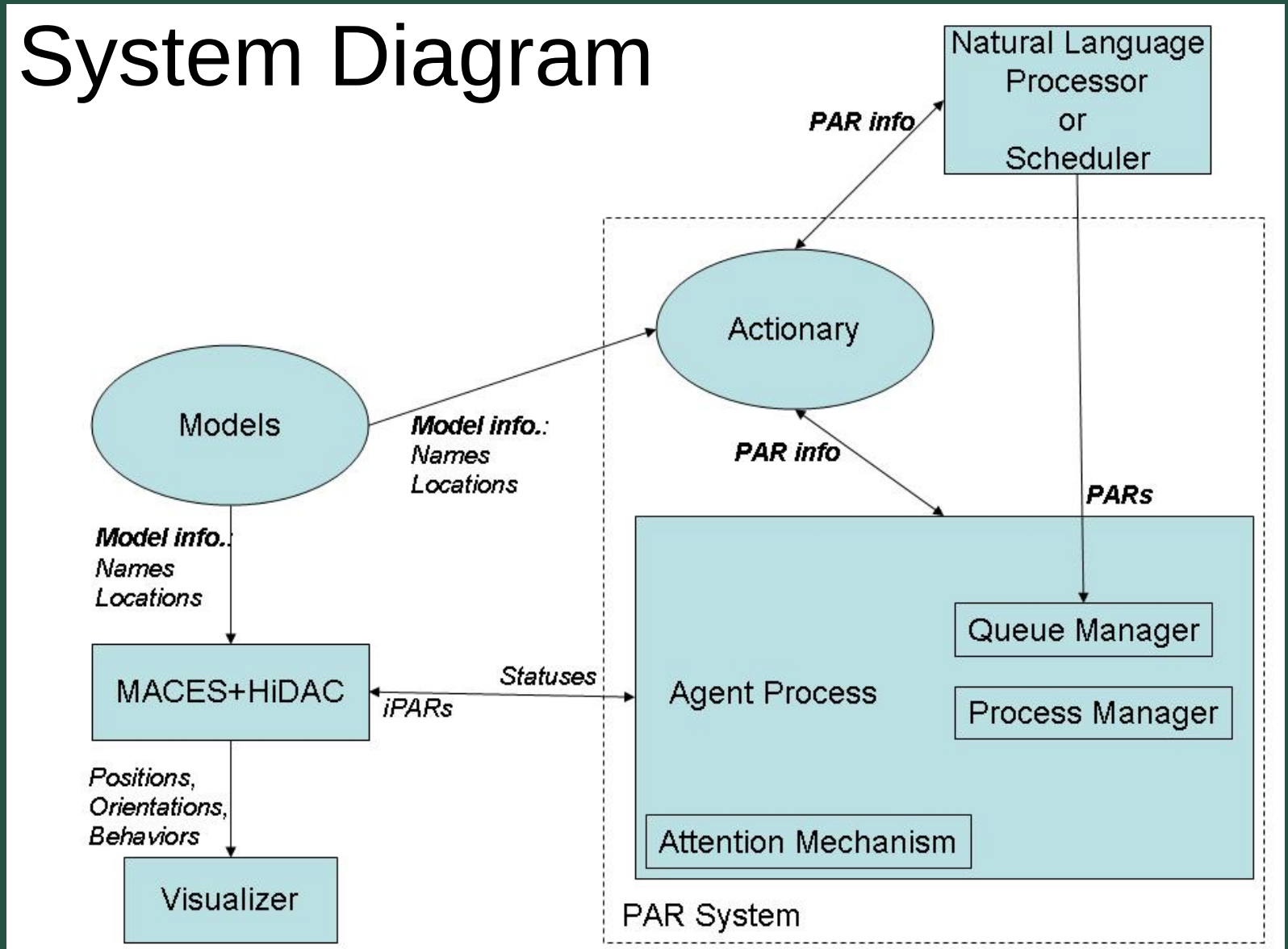
Adding an iPAR to the Actionary

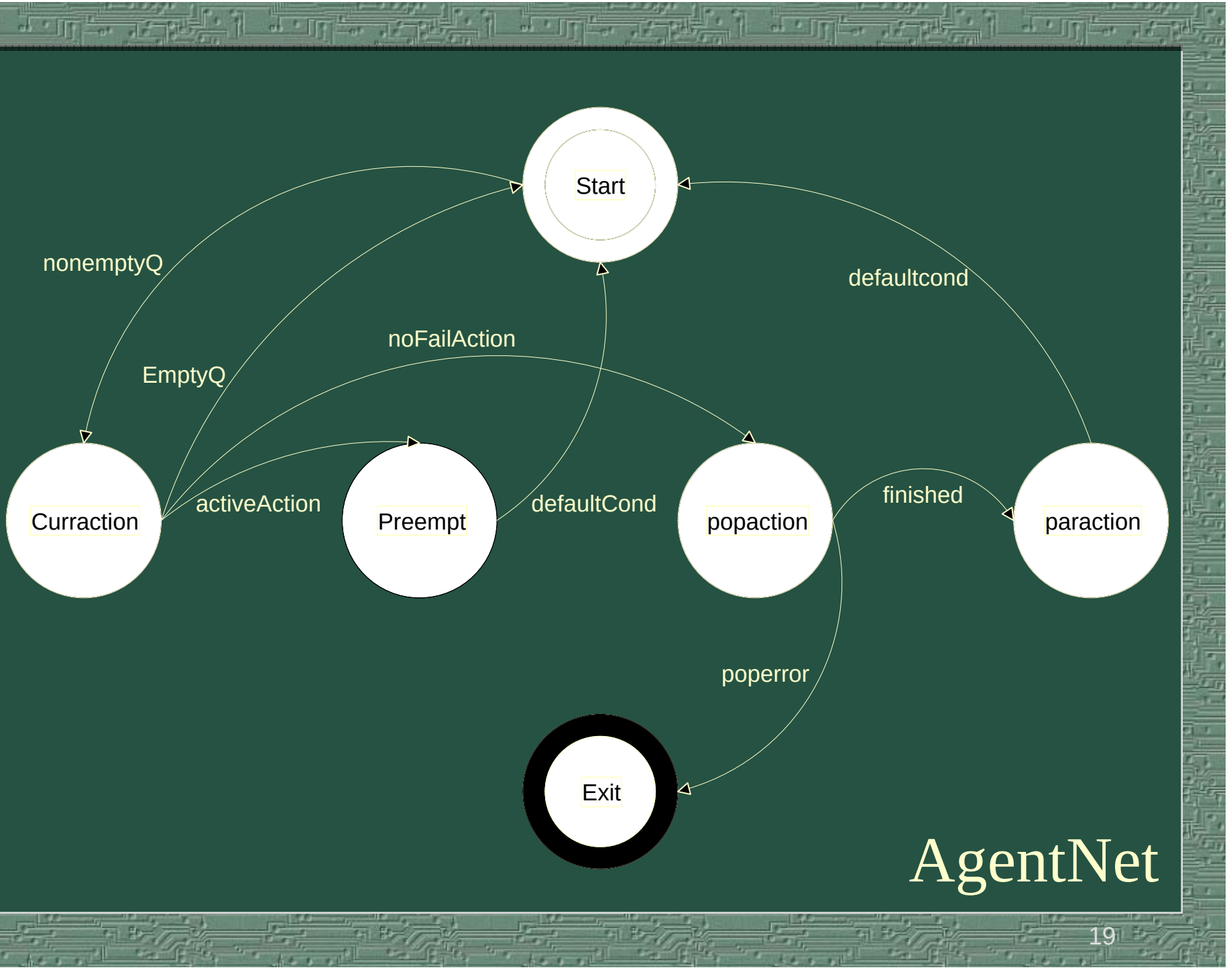
- Determine the corresponding uPAR
- Create new action id
- Fill in name
- Fill in purpose to enable action?
- Fill in act_parent_act_id = uPAR id
- Fill in start time
- Fill in duration if desired
- Alter priority?

Assigning an iPAR to an Agent

```
char* objNames[] = {obj->getObjectName()};

iPAR* ipar = new iPAR(act->getActionName(),
      agent->getObjectName(), objNames);

ipar->setStartTime(partime->getCurrentTime());
ipar->setPriority(11);

// add to agent's queue
agt->addAction(ipar);
```

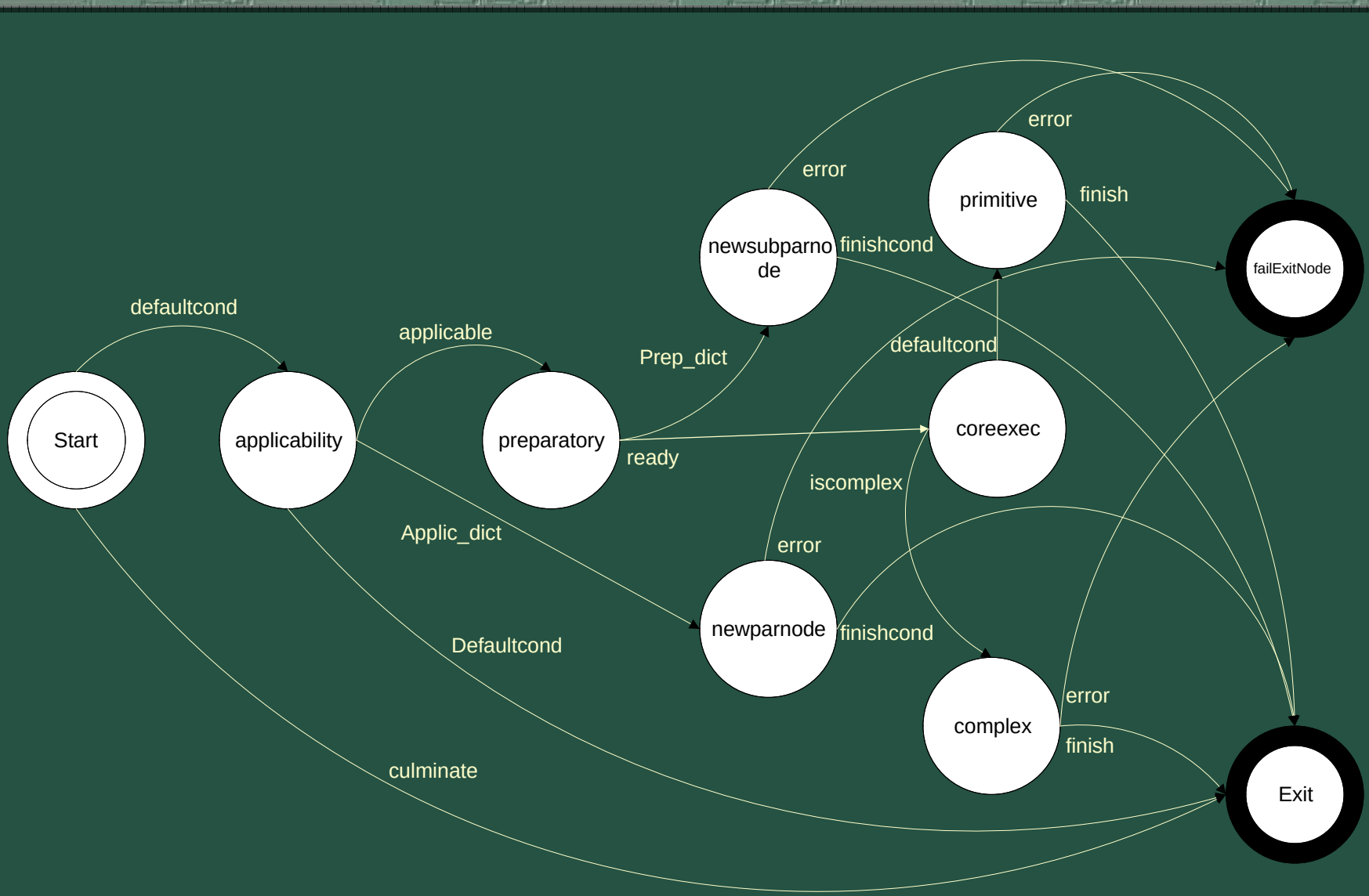
System Diagram




Start
nonemptyQ
EmptyQ
noFailAction
defaultcond
Curraction
activeAction
Preempt
defaultCond
popaction
finished
paraction
poperror
Exit
AgentNet

```
AgentNet::AgentNet(list<iPAR*> *IParQ, AgentProc *ap)
  :LWNet(NUMNODES)
{

  iparQ = IParQ;
  agentproc = ap;

  ipar = 0;
  popError = 0;

  defnormalnode(START, (ACTFUNC)&AgentNet::actfunc_start);
  deftrans(START, (CONDFUNC) &AgentNet::nonemptyQ, CURRACTION);

  defnormalnode(CURRACTION, (ACTFUNC)&AgentNet::actfunc_curraction);
  deftrans(CURRACTION,(CONDFUNC)&AgentNet::emptyQ, START);
  deftrans(CURRACTION,(CONDFUNC)&AgentNet::activeAction, PREEMPT);
  deftrans(CURRACTION,(CONDFUNC)&AgentNet::noFailAction, POPACTION);

  defnormalnode(PREEMPT, (ACTFUNC)&AgentNet::actfunc_preempt);
  deftrans(PREEMPT,(CONDFUNC)&LWNet::defaultcond, START);

  defnormalnode(POPACTION, (ACTFUNC)&AgentNet::actfunc_popaction);
  deftrans(POPACTION,(CONDFUNC)&AgentNet::condfunc_poperror, EXITNODE);
  deftrans(POPACTION, (CONDFUNC)&LWNet::finishcond, PARACTION);

  defnormalnode(PARACTION, (ACTFUNC)&AgentNet::actfunc_paraction);
  deftrans(PARACTION,(CONDFUNC)&LWNet::defaultcond, START);

  defexitnode(EXITNODE);
}
```

AgentNet

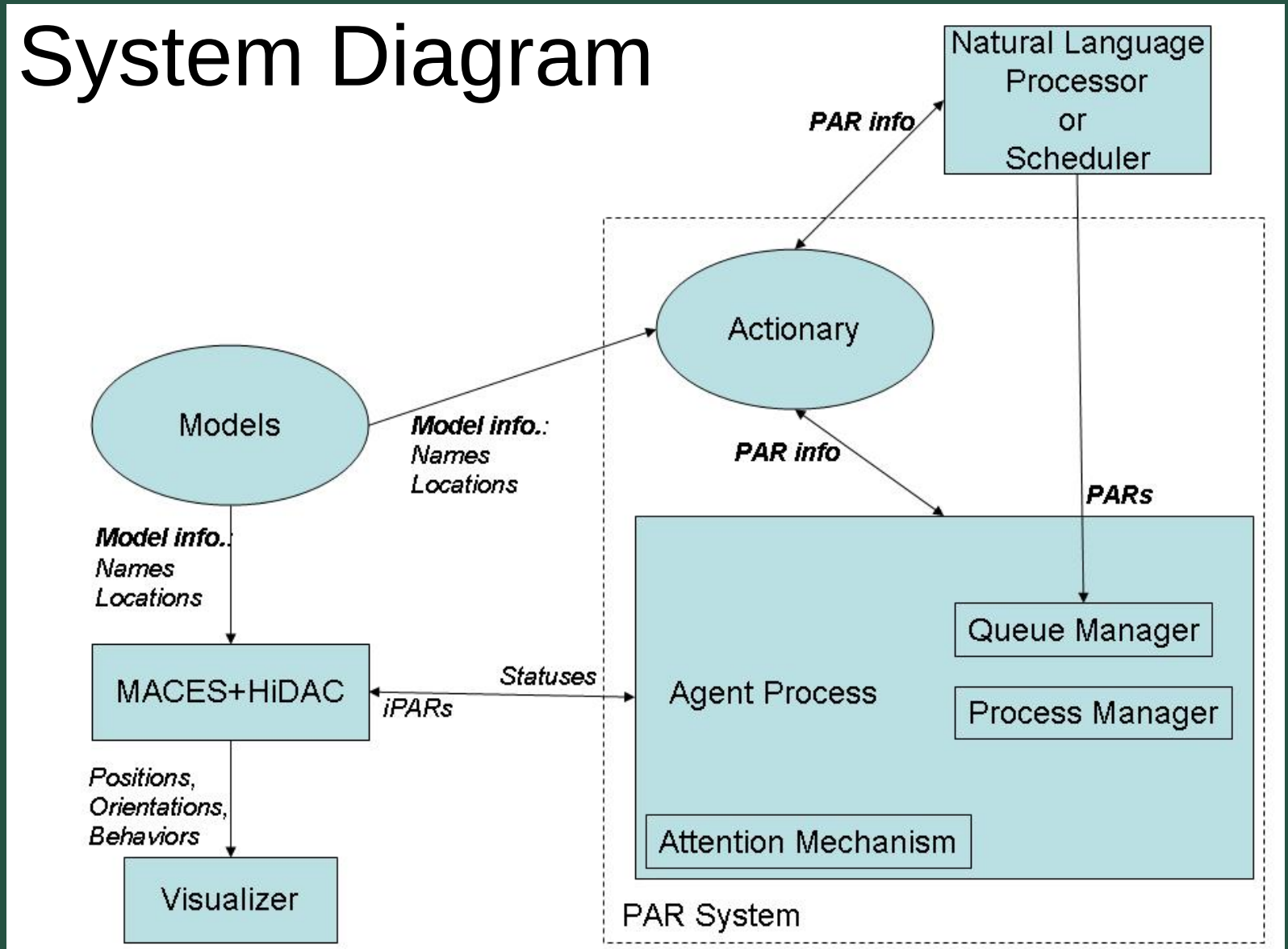

Start
applicability
preparatory
newsubparnode
primitive
failExitNode
coreexec
newparnode
complex
Exit
defaultcond
applicable
Prep_dict
error
finishcond
finish
ready
iscomplex
Applic_dict
Defaultcond
culminate

PARNet

```
defnormalnode(START, (ACTFUNC)&ParNet::actfunc_start);
 deftrans(START, (CONDFUNC)&ParNet::condfunc_culminate, EXITNODE);
 deftrans(START, (CONDFUNC)&LWNet::defaultcond, APPLICABILITY);

 defnormalnode(APPLICABILITY, (ACTFUNC)&ParNet::actfunc_applicability);
 deftrans(APPLICABILITY, (CONDFUNC)&ParNet::condfunc_isapplicdictionary, NEWPARNODE);
 deftrans(APPLICABILITY, (CONDFUNC)&ParNet::condfunc_applicable, PREPARATORY);
 deftrans(APPLICABILITY, (CONDFUNC)&LWNet::defaultcond, EXITNODE);

 defnormalnode(PREPARATORY, (ACTFUNC)&ParNet::actfunc_preparatory);
 deftrans(PREPARATORY, (CONDFUNC)&ParNet::condfunc_isprepdictionary, NEWSUBPARNODE);
 deftrans(PREPARATORY, (CONDFUNC)&ParNet::condfunc_ready, COREEXEC);

 defcallnode(NEWPARNODE, (LWNEW)&ParNet::newpar, NULL, NULL, (void *)app_result);
 deftrans(NEWPARNODE,(CONDFUNC)&LWNet::errorcond,FAILEXITNODE);
 deftrans(NEWPARNODE, (CONDFUNC)&LWNet::finishcond, EXITNODE);

 defcallnode(NEWSUBPARNODE, (LWNEW)&ParNet::newpar,NULL,NULL,&prep_result);
 deftrans(NEWSUBPARNODE,(CONDFUNC)&LWNet::errorcond,FAILEXITNODE);
 deftrans(NEWSUBPARNODE, (CONDFUNC)&LWNet::finishcond, COREEXEC);


 defnormalnode(COREEXEC, (ACTFUNC)&ParNet::actfunc_coreexecution);
 deftrans(COREEXEC, (CONDFUNC)&ParNet::condfunc_iscomplex, COMPLEX);
 deftrans(COREEXEC, (CONDFUNC)&LWNet::defaultcond, PRIMITIVE);

 defcallnode(COMPLEX,(LWNEW)&ParNet::newpar, NULL, (ACTFUNC)&ParNet::postactfunc_complex, &complexobj);
 deftrans(COMPLEX,(CONDFUNC)&LWNet::errorcond,FAILEXITNODE);
 deftrans(COMPLEX, (CONDFUNC)&LWNet::finishcond, EXITNODE);

 defnormalnode(PRIMITIVE, (ACTFUNC)&ParNet::actfunc_primitive,
(ACTFUNC)&ParNet::preactfunc_primitive,(ACTFUNC)&ParNet::postactfunc_primitive);
 deftrans(PRIMITIVE,(CONDFUNC)&LWNet::errorcond,FAILEXITNODE);
 deftrans(PRIMITIVE, (CONDFUNC)&LWNet::finishcond, EXITNODE);

 defexitnode(EXITNODE);
 deffailexitnode(FAILEXITNODE);
```

PARNet

System Diagram



Agent Process

- Inform: message passing
- Perceive: objects that are sensed
- Update:
  - positions, locations, etc
  - Action choices
- Add action: start new performance

Projects/Future Work

- GUIs
- Tools and checkers
- Agents
- Different action types
- Quantifiers
- Properties
- Spatial reasoner
- Maya plug-in for object creating and db

The End

```
void
AgentNet::actfunc_start(void)
{
}
```

Back

```
void
AgentNet::actfunc_curraction(void)
{
}
```

Back

```
void
AgentNet::actfunc_preempt(void)
{
  iPAR* cfipar = this->getFirstAction();
  if (ipar != NULL
     && this->readyToGo(cfipar))
     agentproc->ipt->
          preempt(cfipar->getPriority());
}
```

Back

```
void
AgentNet::actfunc_popaction(void)
{
  iPAR *cfipar = getFirstAction();
  if (cfipar == NULL)
      return;

  if (readyToGo(cfipar))
  {
     ipar = cfipar;
     iparQ->remove(cfipar);
     markfinished();
  }
}
```

Back

```
void
AgentNet::actfunc_paraction(void)
{
  ComplexObj *complexobj =
                      new ComplexObj(ipar,0);
  LWNetList::addnet(new ParNet(complexobj));
}
```

Back

```
bool
AgentNet::nonemptyQ(void)
{
  return (!(iparQ->empty()));
}
```

Back

```
bool
AgentNet::emptyQ(void)
{
  return iparQ->empty();
}
```

Back

```
bool
AgentNet::condfunc_poperror(void)
{
  return popError;
}
```

Back

```
bool
AgentNet::activeAction(void)
{
  return agentproc->ipt->activeAction();
}
```

Back

```
bool
AgentNet::noFailAction(void)
{
  return !(agentproc->ipt->failAction());
}
```

Back

IPAR Table

- List of all IPARs for the agent
- Status:
  - notfound,
  - onqueue,
  - preproc,
  - prepspec,
  - exec,
  - completed,
  - aborted,
  - failure, // set when failure has been detected
  - failed // set after failure recovery

```
void
ParNet::actfunc_start(void)
{
  ap->ipt->setStatus(ipar,preproc);
}
```

Back

```
Bool
ParNet::condfunc_culminate(void)
{
  timer.startTimer("ParNet::condfunc_culminate");
  Bool result;
  PyObject* res;

  res = actionary->testCulminationCond(ipar);
  assert(res);

  if (result = fromPyObjectToBool(res))
      ap->ipt->setStatus(ipar,completed);

  timer.stopTimer("ParNet::condfunc_culminate");
  return result;
}
```

Back

```
void
ParNet::actfunc_applicability(void)
{
  timer.startTimer("ParNet::act_func_applicability");
  app_result = actionary->testApplicabilityCond(ipar);
  assert(app_result);

  if(PyDict_Check(app_result))
  {
    app_dict = 1;
    AgentProc *ap = agentTable.getAgent(ipar->par->getAgent()->getObjectName());
    assert(ap);
    ap->ipt->setStatus(ipar,aborted);
  }
  else if (fromPyObjectToBool(app_result))
  {
     app_status = 1;
  }
  else
    {
      app_status = 0;
      AgentProc *ap = agentTable.getAgent(ipar->par->getAgent()->getObjectName());
      assert(ap);
      ap->ipt->setStatus(ipar,aborted);
    }
  timer.stopTimer("ParNet::act_func_applicability");
}
```

Back

```
Bool
ParNet::condfunc_applicable(void)
{
  return app_status;
}
```

Back

```
Bool
ParNet::condfunc_isapplicdictionary(void)
{
  return app_dict;
}
```

Back

```
LWNet *
ParNet::newpar(void *args)
{
  timer.startTimer("ParNet::newpar");

  ComplexObj **cobj = (ComplexObj **)args;
  assert(cobj);

  ActionNet *actionnet = new ActionNet(*cobj);
  assert(actionnet);

  timer.stopTimer("ParNet::newpar");
  return actionnet;
}
```

Back

```
void
ParNet::actfunc_preparatory(void)
{
   timer.startTimer("ParNet::actfunc_preparatory");

   ap->ipt->setStatus(ipar,prepspec);
   prep_result = actionary->testPreparatorySpec(ipar);
   assert(prep_result);

   if(PyDict_Check(prep_result)) {
      prep_dict = 1;  }
   else if (fromPyObjectToBool(prep_result)) {
      prep_status = 1; }
   else { prep_status = 0; }

   timer.stopTimer("ParNet::actfunc_preparatory");
}
```

Back

```
Bool
ParNet::condfunc_ready(void)
{
  return prep_status;
}
```

Back

```
Bool
ParNet::condfunc_isprepdictionary(void)
{
  return prep_dict;
}
```

Back

```
void
ParNet::actfunc_coreexecution(void)
{
   timer.startTimer("ParNet::actfunc_coreexecution");
   ap->ipt->setStatus(ipar,exec);
   PyObject *complexpar  = actionary->testExecutionSteps(ipar);
   assert(complexpar);
   complexobj->pyobj = complexpar;
   complexobj->complexParent = ipar;

   if(PyDict_Check(complexpar)) {
      prep_dict = 1; }

   if(PyDict_GetItemString(complexpar,"COMPLEX")) {
      complex = 1;  }
   else if (PyDict_GetItemString(complexpar,"PRIMITIVE")) {
      complex = 0;  }
   timer.stopTimer("ParNet::actfunc_coreexecution");
}
```

Back

```
Bool
ParNet::condfunc_iscomplex(void)
{
  return complex;
}
```

Back

```
void
ParNet::preactfunc_primitive(void)
{
  if(!(callback =
    actionTable.getFunctions(ipar->par->getActionName())))
    printf("ERROR: No execution function found for action
%s\n",ipar->par->getActionName());
  else
    callback->func(ipar);
}
```

Back

```
void
ParNet::actfunc_primitive(void)
{
  FailData *failData = new FailData;
  if(callback)
    {
      switch(callback->finished(ipar, &failData))  {
        case SUCCESS:
           markfinished();
           break;
        case FAILURE:
           markerror();    // flag this node as error
           if(ipar->complexParent) {
               ap->ipt->setStatus(ipar->complexParent,failure);
               failData->complexIPAR = ipar->complexParent;  }
            ap->inform("Failure",(void *)failData);
           break;
        default:
           break;
     }
  }
  else markfinished();
}
```

Back

```
void
ParNet::postactfunc_primitive(void)
{
    if((ap->ipt->getStatus(ipar) != failure) && \
       (ap->ipt->getStatus(ipar) != failed) &&
          (ap->ipt->getStatus(ipar) != aborted))
      {
       ap->ipt->setStatus(ipar,completed);
      }
}
```

Back

```
void
ParNet::postactfunc_complex(void)
{
   if(!errorcond())
    {
         ap->ipt->setStatus(ipar,completed);
    }
}
```

Back