xUML to AADL
Purpose of Translation

Analyze the runtime characteristics of a model expressed in xUML

Improve runtime structure

Options:

- Active object == thread (logical thread) & thread optimization to OS threads
- Define task architecture (OS threads) & active object -> thread mapping
Sample Domain

Sensor Data Processing

myCounterpartAirTrack = this -> CPR1
$USE TM
generate AT1.updateAirTrack(currentLatitude, currentLongitude, currentAltitude) to myCounterpartAirTrack
$ENDUSE

Bridge

Track Management
Mapping Domains

Domains are mapped to packages in AADL
Every definition in the public section

```plaintext
package xUMLBasicTypes
public
...
end xUMLBasicTypes;
package SensorDataProcessingDomain
public
...
end SensorDataProcessingDomain;
package TrackManagementDomain
public
...
end TrackManagementDomain;
```
Finding xUML Threads

Two Sources

- External Device Stimuli
- State Machines

In Our Example

- Active Sensor Hardware
- AirTrack State Machine
Communication Semantics

xUML Queuing semantics

- Event Data Communication between threads
AADL Threads for Example

thread ActiveSensorThread
features
  createActiveSensorTrackReport: in event data port CreateActiveSensorTrackEvent;
  initializeAirTrack: out event data port TrackManagementDomain::InitializeAirTrackEvent;
  updateAirTrack: out event data port TrackManagementDomain::UpdateAirTrackEvent;
  deleteAirTrack: out event data port TrackManagementDomain::DeleteAirTrackEvent;
end ActiveSensorThread;

thread AirTrackThread
features
  initializeAirTrack: in event data port InitializeAirTrackEvent;
  updateAirTrack: in event data port UpdateAirTrackEvent;
  deleteAirTrack: in event data port DeleteAirTrackEvent;
end AirTrackThread;
Implicit Object Management

Objects are assumed to be managed by its class in xUML

- Find objects
- Manage object memory for creation/deletion

Need to be explicit in AADL

- In the form of “Collection”
data ActiveSensorTrackReport
features
initialize: subprogram InitializeActiveSensorTrackReportInstance;
update: subprogram UpdateActiveSensorTrackReportInstance;
delete: subprogram DeleteActiveSensorTrackReportInstance;
end ActiveSensorTrackReport;
data ActiveSensorTrackReportCollection
features
find: subprogram FindActiveSensorTrackReportCollection;
create: subprogram CreateActiveSensorTrackReportCollection;
delete: subprogram DeleteActiveSensorTrackReportCollection;
update: subprogram UpdateActiveSensorTrackReportCollection;
end ActiveSensorTrackReportCollection;
Call Sequences

thread implementation ActiveSensorThread.impl

subcomponents
  reportCollection: data ActiveSensorTrackReportCollection;
  activeSensorTrackCollection: data ActiveSensorTrackCollection;
  airTrackCollection: data AirTrackCollection;

calls
  createReport: { find1: subprogram ActiveSensorTrackReportCollection.find;
    create1: subprogram ActiveSensorTrackReportCollection.create;};
  updateReport: { find2: subprogram ActiveSensorTrackReportCollection.find;
    update1: subprogram ActiveSensorTrackReportCollection.update;};
  deleteReport: { find3: subprogram ActiveSensorTrackReportCollection.find;
    delete1: subprogram ActiveSensorTrackReportCollection.delete;};

connections
  c1: event data port create1.initializeAirTrack->initializeAirTrack;
  c2: event data port update1.updateAirTrack->updateAirTrack;
  c3: event data port delete1.deleteAirTrack->deleteAirTrack;
  p1: parameter createActiveSensorTrackReport->find1.report;
  p2: parameter createActiveSensorTrackReport->create1.report;
  p3: parameter createActiveSensorTrackReport->find2.report;
  p4: parameter createActiveSensorTrackReport->update1.report;
  p5: parameter createActiveSensorTrackReport->find3.report;
  p6: parameter createActiveSensorTrackReport->delete1.report;

end ActiveSensorThread.impl;
Final System

process TrackingProcess
features
    createActiveSensorTrackReport : in event data port SensorDataProcessingDomain::CreateActiveSensorTrackEvent;
end TrackingProcess;
process implementation TrackingProcess.Impl
subcomponents
    sensorThread: thread SensorDataProcessingDomain::ActiveSensorThread;
    airTrackThread: thread TrackManagementDomain::AirTrackThread {xUML::Multiplicity => 100;};
connections
    c1: event data port sensorThread.initializeAirTrack- >airTrackThread.initializeAirTrack {xUML::Connection_Multiplicity => OneToOne;};
    c2: event data port sensorThread.updateAirTrack- >airTrackThread.updateAirTrack {xUML::Connection_Multiplicity => OneToOne;};
    c3: event data port sensorThread.deleteAirTrack- >airTrackThread.deleteAirTrack {xUML::Connection_Multiplicity => OneToOne;};
    c4: event data port createActiveSensorTrackReport- >sensorThread.createActiveSensorTrackReport;
end TrackingProcess.Impl;

device ActiveSensorDevice
features
    createActiveSensorTrackReport: out event data port SensorDataProcessingDomain::CreateActiveSensorTrackEvent;
end ActiveSensorDevice;
processor MyProcessor
end MyProcessor;
system Final
end Final;
system implementation Final.Impl
subcomponents
    sensor: device ActiveSensorDevice;
    proc: processor MyProcessor;
    trackProcess: process TrackingProcess;
connections
    c1: event data port sensor.createActiveSensorTrackReport- >trackProcess.createActiveSensorTrackReport;
end Final.Impl;
To Add in an AADL tool (OSATE)

End-to-end latency requirements

Periodicity of events, both external (e.g. sensor interrupts) and internal (timers – could be extracted from the xUML model)

Execution time of subprograms

Processor Speed

Network Speed
Translation of Message Semantics

xUML Semantics

- Closed Blocking. This represents a function call where the caller can send data and expects and waits for an answer from the callee before continuing its execution.
- Closed Non-Blocking. In this case the caller also expects an answer but it will not wait to get it before continuing its execution. Instead it queries for the answer at a later time.
- Open. This involves a transfer of data from the caller to the callee. The caller does not wait for the completion of the callee neither expects any answer from it.

AADL Semantics

- Closed Blocking. In this case the callee is a subprogram and the message from the caller to the callee a subprogram call.
- Closed Non-Blocking. In this case the callee is a thread (and hence the caller is another thread). The message is a data port connection from caller to callee and an event port connection from the callee to the caller to notify the completion of the execution.
- Open. In this case the caller and the callee are both thread and the message is only a data port connection from the caller to the callee.
Questions?