A++: a collection of C++ idioms for activity planning and sequencing

Pierre F Maldague

Contents

Executive Summary

- GOAL

  Answer Seung Chung's question: "Can you do APGen adaptation in C++?"

- METHOD

  Provide C++ idioms that resemble the AAF language in style and meaning

- HOPES

  1. Achieve (significantly?) better performance than APGen
  2. Minimize danger that present-day adapters will run away screaming when they see the A++ version of their beautiful AAFs

- RESULTS

  The next slides show a summary of what was accomplished

RESULTS (1): Adaptation – activity resource usage

AAF:

```
resource usage
        step: duration default to 00:30:00;
        every_time: time default to start;
        while(every_time <= finish) {
                use ampMgr(current, Duration, phase, delay_1) at every_time;
                every_time = every_time + step;
        }
```

A++:

```
void real_activity::model() {
        cppDuration step("00:30:00");
        for(cppTime every_time = start; every_time <= finish; every_time += step) {
                AbsResEvent<ampMgr>().use(ampMgr::data(current, Duration, phase, delay_1)).at(every_time);
        }
}
```

RESULTS (2): Adaptation – activity modeling

AAF:

```
Modeling
        use Power( -power ); wait for 00:30:00;
        use Power( power ); wait for 01:30:00;
        use Power( -power ); wait for 00:30:00;
        use Power( power );
```

A++:

```
void threaded_act::modeling() {
        ResEvent<float_resource>( Power ). use ( -power ). ASAP(); wait_for( cppDuration( "00:30:00” ));
        ResEvent<float_resource>( Power ). use ( power ). ASAP(); wait_for( cppDuration( "01:30:00” ));
        ResEvent<float_resource>( Power ). use ( -power ). ASAP(); wait_for( cppDuration( "00:30:00” ));
        ResEvent<float_resource>( Power ). use ( power ). ASAP();
}
```

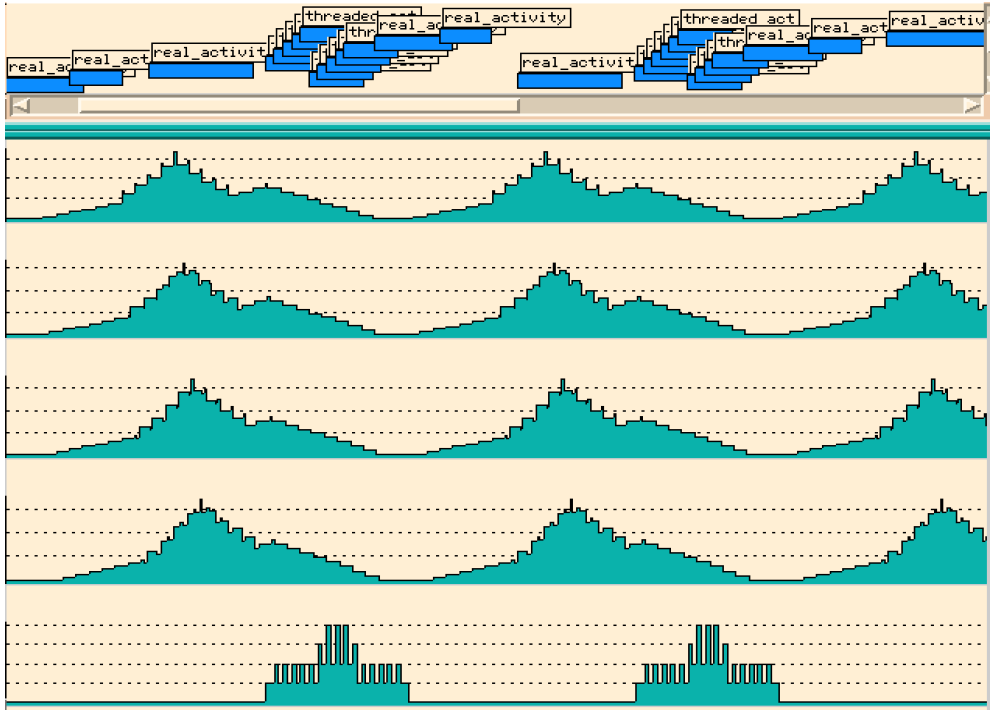
RESULTS(3): Testing, Measuring Performance

AAF process

AAF
- Concrete resources
- Abstract resources
- Real_activity
- Threaded_activity
- Apf activity
  Creates 4000 sub-activities

Modeling

102 sec.



40 sec.

XMLTOL

A++ process

Main.C, Adaptation.C, Framework.C

- Concrete resources classes
- Abstract resources classes
- Activity classes
- Apf function
  Creates 4000 sub-activities

Model_intfc::run()

4.2 sec.

Internal Data
- Activity Instances
- Event Queue
- Resource histories

Write_xmltol()

22 sec.

XMLTOL

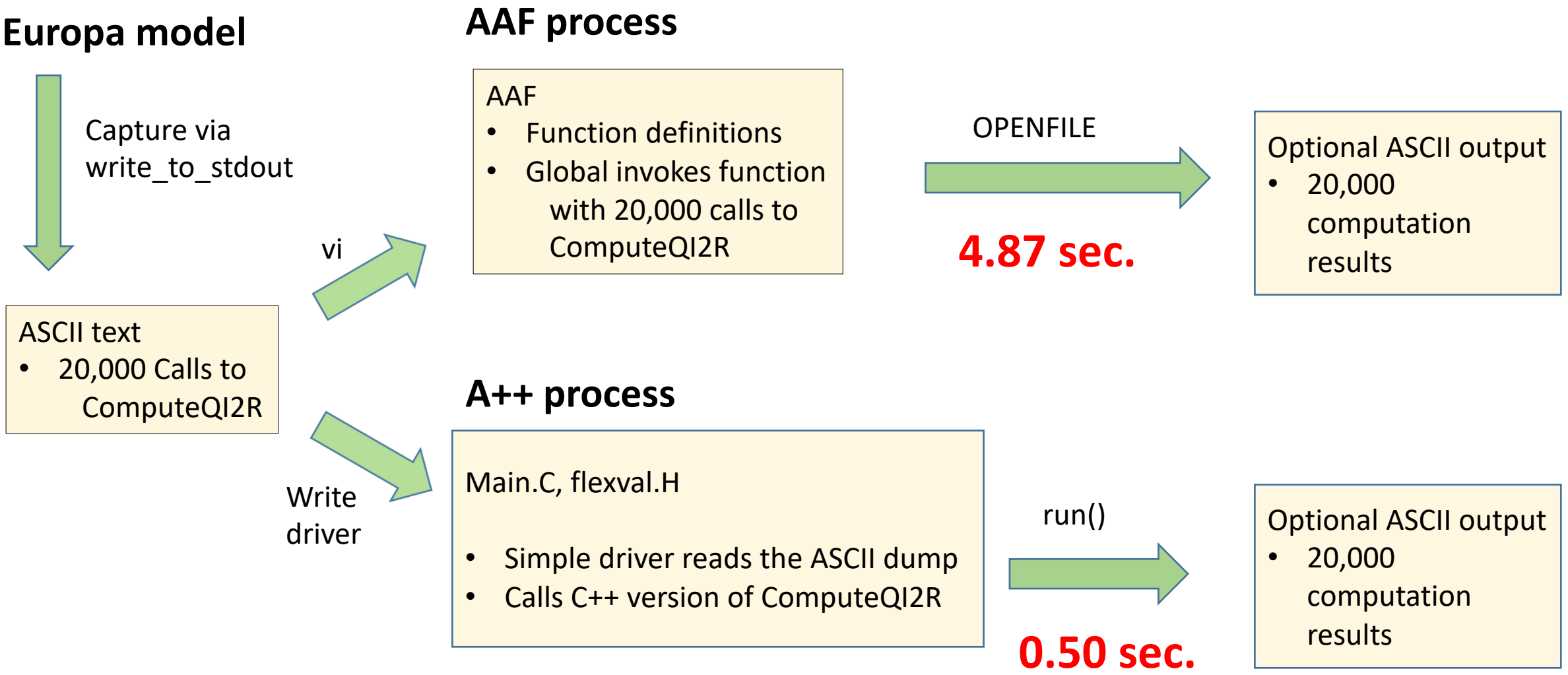
OPENFILE

RESULTS(4): AAF vs. A++ function (from Europa)

```
function euler0(DCM)
   parameters
      DCM: array default to [[0.0,0.0,0.0],[0.0,0.0,0.0],[0.0,0.0,0.0]] ; {
      q: array default to [0.0,0.0,0.0,0.0] ;
      TC: float default to 0.0 ;
      QM: array default to [0.0,0.0,0.0,0.0] ;
      X: float default to 0.0 ;
      i: integer default to 0 ;
      ID0: integer default to 0 ;
      qn: float default to 0.0 ;
      TC = DCM[0][0] + DCM[1][1] + DCM[2][2] ;
      QM[0] = sqrt(fabs(2.0 * DCM[0][0] + 1.0 - TC) / 4.0) ;
      QM[1] = sqrt(fabs(2.0 * DCM[1][1] + 1.0 - TC) / 4.0) ;
      QM[2] = sqrt(fabs(2.0 * DCM[2][2] + 1.0 - TC) / 4.0) ;
      QM[3] = sqrt(fabs(1.0 + TC) / 4.0) ;
      X = QM[0] ;
      ID0 = 0 ;
      i = 1 ;
      while( i <= 3) {
         if( X < QM[i]) {
            X = QM[i] ;  ID0 = i ;
         }
         i = i + 1 ; }
```

```
flexval euler0(
                flexval DCM
                ) {
      flexval q; // array default to [0.0,0.0,0.0,0.0] ;
      double TC = 0.0 ;
      flexval QM; // array default to [0.0,0.0,0.0,0.0] ;
      double X = 0.0 ;
      long int i = 0 ;
      long int ID0 = 0 ;
      double qn = 0.0 ;
      TC = DCM[0][0] + DCM[1][1] + DCM[2][2] ;
      QM[0] = sqrt(fabs(2.0 * DCM[0][0] + 1.0 - TC) / 4.0) ;
      QM[1] = sqrt(fabs(2.0 * DCM[1][1] + 1.0 - TC) / 4.0) ;
      QM[2] = sqrt(fabs(2.0 * DCM[2][2] + 1.0 - TC) / 4.0) ;
      QM[3] = sqrt(fabs(1.0 + TC) / 4.0) ;
      X = QM[0] ;
      ID0 = 0 ;
      i = 1 ;
      while( i <= 3) {
         if( X < (double)QM[i]) {
            X = QM[i] ; ID0 = i ;
         }
         i = i + 1 ;  }
```

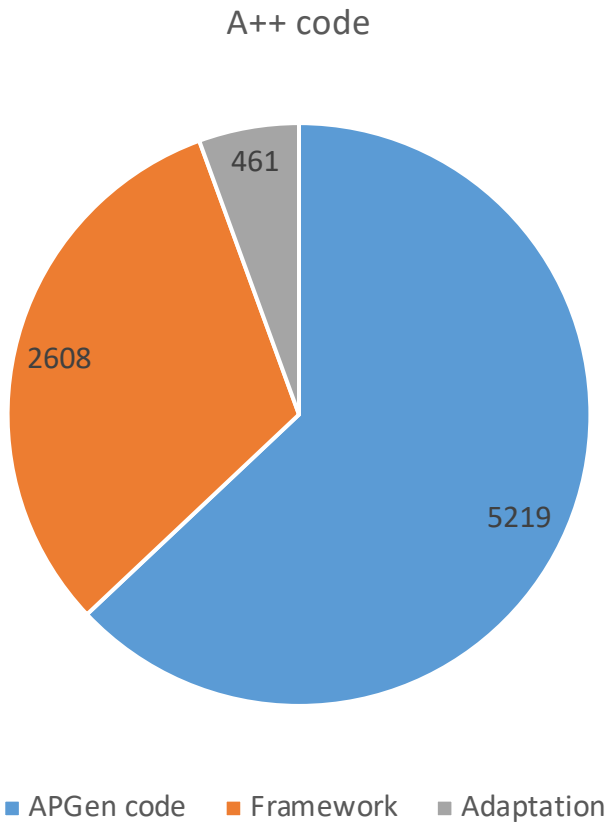
RESULTS(5): Testing, Measuring Performance



A++ Overview: “a collection of C++ idioms”

- What’s an idiom?
- Google:
  1. A group of words established by usage as having a meaning not deducible from those of the individual words (e. g., rain cats and dogs, see the light).
  2. A characteristic mode of expression in music or art.
- Both definitions are appropriate:
  - The collection of A++ idioms, together with foundation classes adapted from APGen, provide a powerful planning and simulation capability that is not obvious in the underlying C++.
  - The idioms provide strong hints about how to implement specific behaviors, based on experience with APGen adaptation – which may not be music but definitely qualifies as art or craft.

A++ Overview: Code composition



A++ Overview: classes for concrete resources

```
// Resource Classes
class amps: public float_resource {
   public: amps(string s, resource_array_base& P) : float_resource(s, P) {}
           void use(double x) {
                   float_resource::use(x);
           }
   };
class power: public float_resource {
   public: power() : float_resource("power") {}
           void use(double x) {
                   float_resource::use(x);
           }
  };
```

A++ Overview: classes for abstract resources (1)

- After some trial and error, I discovered three important design principles for A++:

1. **Keep data separate from methods**
2. **Make the methods fully generic**
3. **Collect methods in a pure virtual class template**

- Abstract resources become a template with the class containing the data (parameters, local variables) as a parameter

A++ Overview: classes for abstract resources (2)

Template definition for abstract resources

```
template <class ext_data>
  class absRes {
  public:
        typedef ext_data data;
        data      myData;
        absRes(data D)
              : myData(D)
              {}
        virtual void use() = 0;
  };
```

A++ Overview: classes for abstract resources (3)

An example

```
class ampMgrData {
   public:
            ampMgrData() : current(0), duration_of_usage("00:00:00"), delay_between_uses("00:00:00") {}
            ampMgrData(double x, cppDuration y, string z, cppDuration w)
                        :       current(x), duration_of_usage(y),
                                which_phase(z), delay_between_uses(w) {}
            double              current;
            cppDuration         duration_of_usage;
            string              which_phase;
            cppDuration         delay_between_uses;
   };
class ampMgr: public absRes<ampMgrData> {
   public:
            ampMgr(const ampMgrData& D) : absRes<ampMgrData>(D) {}
            void        use();
   };
```

The only part of the definition that is not boilerplate

A++ Overview: classes for activities (1)

- Two basic activity templates:
  - activity_base for activity types that utilize the “resource usage” style of modeling (similar to SEQGen’s “ground-expanded activities”)
  - activity_thread for activity types that utilize the “modeling” style of modeling (similar to SEQGen’s use of execution engines)
- Both templates provide the standard activity attributes:

  type, id, start, span, finish, Color, subsystem, legend, pattern, name, baggage
- The adapter must provide the following:
  - Activity parameters
  - model() or modeling() functions that mimic the AAF’s resource usage or modeling section
  - A method that expresses parameters as an APGen-like “flexval array”

A++ Overview: classes for activities (2) - example

```
class threaded_act: public activity_thread {
        cppDuration     Duration;
        double          power;
        threaded_act(   const string& Name, const cppTime&   start_time,
                        const cppDuration& param_1, const double& param_2)
            : activity_thread(Name, "threaded_act", start_time, param_1),
                Duration(param_1), power(param_2) {}
        threaded_act(const threaded_act& A)
            : activity_thread(A), Duration(A.Duration), power(A.power) {}
        ~threaded_act() {}
        flexval     get_parameters() const {
                    flexval V, W;
                    W["name"] = "Duration"; W["value"] = Duration;
                    V[0] = W;
                    W["name"] = "power"; W["value"] = power;
                    V[1] = W;
                    return V;
        }
        void                modeling();
};
```

The only part of the definition that is not boilerplate

This method allows the activity to export its parameters e. g. to an activity editor

A++ Overview: beyond the AAF language (1)

In A++ you can do things that cannot be done in the AAF, like an instance of an abstract resource destroying itself when it is no longer needed:

```
void Res::ampMgr::use() {
   cppTime                                S(model_intfc::now());
   for(int k = 0; k < 10; k++) {
            AbsResEvent<start_using>().use(start_using::data(k, myData.current)).at(S);
            AbsResEvent<stop_using>().use(stop_using::data(k, myData.current)).at(S + myData.duration_of_usage);
            S = S + myData.delay_between_uses;
   }
   cppDuration margin("00:01:00");
   cppTime time_of_death = S + myData.duration_of_usage + margin;
   AbsResEvent<kill_mgr>().use(this).at(time_of_death);
}
```

A++ Overview: beyond the AAF language (2)

- You can introduce new adaptation style – e. g. an “operator/assistant” to whom you delegate a “list of things to do”
- Future possibility: define templates to specify “goals to achieve” and link the things to do to the goals

```
void Res::ampMgr::use() {
        cppTime                         start(model_intfc::now());
        for(int k = 0; k < 10; k++) {
                thingsForOpToDo.add(&Op::start_using, Op::data(k)).at(start);
                thingsForOpToDo.add(&Op::stop_using, Op::data(k)).at(start + myData.duration_of_usage);
                start = start + myData.delay_between_uses;
        }
        cppDuration margin("00:01:00");
        cppTime time_of_death = start + myData.duration_of_usage + margin;
        thingsForOpToDo.add(&Op::kill_mgr, Op::data(-1)).at(time_of_death);
        thingsForOpToDo.schedule_everything(); // including my own demise
}
```

Where to go from here

Things that naturally come to mind:

- Implement mechanism for importing A++ resources into APGen
- Modify the APGen event loop so it can include A++ events
- Convert the basic Europa schedulers (DSN allocations, Telecom activities) to an A++ implementation; this would allow potential enthusiasts like Eric Ferguson and Mike Schaffer to provide feedback on A++
- Extend the IMCE ontology so it can capture the behavior expressed in the A++ adaptation templates
- Extend the A++ methodology to an SMF++ framework for SEQGen