Appendix A

UML Grammar

The grammar presented in Figure A.1 is for the data definition section of the UML. It is an extract from the actual `yacc` description used to implement the interpreter in the prototype.

```
/* Interpreter directives */

%token                  D_INSERT
%token                  D_REPLACE
%token                  D_DELETE


/* Keywords */

%token                  UNIVERSE
%token                  ELEMENT
%token                  PROPERTY
%token                  FUNCTION
%token                  VFUNCTION
%token                  CONVERT
%token                  CONSTANT
%token                  ENTITY


/* Primitive types and their literals */

%token                  INTEGER
%token <integer>        LINTEGER
%token                  REAL
%token <real>           LREAL
%token                  STRING
%token <string>         LSTRING
%token                  BOOLEAN


/* Name of a Universe, Element, variable, etc... */

%token <string>         NAME
```

```
/* Comment */

%token <string>         COMMENT


/* Constants */

%token                  C_FALSE
%token                  C_TRUE


/* Code constructs */

%token                  VAR
%token                  FOR
%token                  IF
%token                  ELSE
%token                  WITH

%token                  EQUIV_OP

%%


file    :       /* Nothing */
        |       file file_component
        ;

file_component: universe
        |       element
        |       constant_ext
        |       property_ext
        |       function_def
        |       entity_def
        |       D_INSERT
        |       D_REPLACE
        |       D_DELETE comp_ident dot_name
        ;

comp_ident:     UNIVERSE
        |       ELEMENT
        |       CONSTANT
        |       FUNCTION
        |       CONVERT
        |       PROPERTY
        |       ENTITY
        ;

universe:       UNIVERSE name_def '{' univ_body '}'
        ;

univ_body:      /* Nothing */
        |       univ_body univ_def
        ;

univ_def:       constant
        |       element
        |       property
        |       converter
        |       function
        ;

constant_ext:   CONSTANT ext_var_decl '=' initialiser ';'
        ;
```

```
constant:       CONSTANT var_decl '=' initialiser ';'
        ;

initialiser:    literal
        |       '[' literal_list ']'
        ;

literal:        LREAL
        |       LINTEGER
        |       LSTRING
        |       boolean
        ;

literal_list:   literal_list ',' literal
        |       literal
        ;

boolean:        C_FALSE
        |       C_TRUE
        ;

element:        element_decl
        |       element_def
        ;

element_def:    ELEMENT elemname '{' elem_def_body '}'
        ;

element_decl:   ELEMENT elemname ';'
        ;

elemname:       dot_name ':' NAME
        |       dot_name
        ;

elem_def_body:  /* Nothing */
        |       elem_def_body elem_def
        ;

elem_def:       constant
        |       element
        |       property
        |       converter
        |       function
        ;

property:       PROPERTY var_decl ';'
        ;

property_ext:   PROPERTY ext_var_decl ';'
        ;

converter:      converter_def
        |       converter_decl
        ;

converter_decl: CONVERT NAME ';'
        ;

converter_def:  CONVERT NAME '{' '}'
        ;

function:       function_decl
        |       function_def
        ;
```

```
function_decl:  FUNCTION function_proto ';'
        |       VFUNCTION function_proto ';'
        ;

function_def:   FUNCTION function_proto code_block
        |       VFUNCTION function_proto code_block
        ;

function_proto: dot_name return_type
        |       dot_name param_list return_type
        ;

param_list:     '(' ')'
        |       '(' var_decl_list ')'
        ;

var_decl_list:  var_decl_list ',' var_decl
        |       var_decl
        ;

var_decl:       NAME ':' type_decl
        ;

ext_var_decl:   dot_name ':' type_decl
        ;

return_type:    /* Nothing */
        |       ':' type_decl
        ;

code_block:     '{' code_block_body '}'

code_block_body:        /* Nothing */
        |               code_block_body code_statement
        ;

code_statement: variable_decl
        ;

variable_decl:  VAR var_decl ';'
        ;

entity_def:     ENTITY NAME ':' NAME '{' entity_def_body '}'
        ;

entity_def_body:        /* Nothing */
        |               entity_def_body function
        ;


/* Name definition which may involve inheritance */

name_def:       NAME ':' dot_name
        ;

dot_name:       dot_name '.' NAME
        |       NAME
        ;
```

```
/* Type declaration */

type_decl:      primitive
        |       primitive list_decl
        |       NAME
        |       NAME list_decl
        ;

primitive:      REAL
        |       INTEGER
        |       STRING
        |       BOOLEAN
        ;

list_decl:      '[' LINTEGER ']'
        |       '[' ']'
        ;
```

Figure A.1 `yacc` description of UML grammar.

Appendix B

Benchmark Manager, Entity & UML Source Code

The simulation execution benchmarks described in section 6.5 used the minimalistic UML definition shown in Figure B.1. Those tests that monitored two components required the

```
UNIVERSE Benchmark
{
    PROPERTY    quanta : INTEGER;

    VFUNCTION   Construct;
    VFUNCTION   Update;
    VFUNCTION   Destruct;
}

ENTITY bmark1 : Benchmark
{
    FUNCTION Construct;
    FUNCTION Update;
    FUNCTION Destruct;
}
```

Figure B.1 UML definition used in the prototype evaluation.

introduction of a second property: `quanta2`.

The source code for the benchmark entity is shown in Figure B.2 and the source code for the manager used to monitor the state updates in given in Figure B.3.

```
#include "ENT.h"


bool    EmbedFunctions( int noofParams, ... );

bool    Construct( void );
bool    Update( void );
bool    Destruct( void );


static  ENT             *ent = NULL;
static  UMLProperty     *quanta;
#ifdef TWO_PROPS
static  UMLProperty     *quanta2;
#endif // end TWO_PROPS
static  UMLInstance     *quantaInst;


int main( int argc, char *argv[] )
{
    if ( argc != 3 || argv[1][0] != '-' )
    {
        cerr << "usage: " << argv[0] << " [-qt] entity_name\n";
        return ( 1 );
    }

    Process::IPCipc = Process::IPC::IPC_NONE;

    if ( argv[1][1] == 'q' )
    {
        ipc |= Process::IPC::IPC_QNX;
    }
    if ( argv[1][1] == 't' )
    {
        ipc |= Process::IPC::IPC_TCPIP;
    }

    try
    {
        ent = new ENT( ipc, argv[2] );

        (void)ent->callback( MSG_UML_INIT_DEF, EmbedFunctions );

        ent->serviceEvents();

        delete ent;
    }
    catch ( ENTCTORError )
    {
        cerr << argv[0] << ": construction failed... terminating\n";
        return ( 1 );
    }
    catch ( ENTError )
    {
        delete ent;
        cerr << argv[0] << ": terminating\n";
        return ( 1 );
    }

    return ( 0 );
}
```

```
bool EmbedFunctions( int noofParams, ... )
{
    UMLEntity   *entity;

    if ( (entity = ent->definition()) == NULL )
    {
        cerr << "EmbedFunctions: can't locate entity definition\n";
        return ( false );
    }


    UMLFunction *construct, *destruct, *update;


    construct = entity->findFunction( "Construct" );
    destruct = entity->findFunction( "Destruct" );
    update = entity->findFunction( "Update" );

    if ( construct == NULL || destruct == NULL || update == NULL )
    {
        cerr << "EmbedFunctions: can't locate functions\n";
        return ( false );
    }

    construct->setCode( Construct );
    update->setCode( Update );
    destruct->setCode( Destruct );

    return ( true );
}


bool Construct( void )
{
    UMLCompType          compType;
    UMLInstance::ID      id = ent->instanceID();

    if ( (compType = ent->definition()->find( "quanta",
                (UMLComponent *&)quanta, id )) != UML_C_PROPERTY )
    {
        cerr << "ERROR: can't locate property\n";
        return ( false );
    }

    if ( (quantaInst = quanta->instance( id )) == NULL )
    {
        cerr << "ERROR: can't locate instance of property\n";
        return ( false );
    }

#ifdef TWO_PROPS
    if ( (compType = ent->definition()->find( "quanta2",
                (UMLComponent *&)quanta2, id )) != UML_C_PROPERTY )
    {
        cerr << "ERROR: can't locate property\n";
        return ( false );
    }
#endif // end TWO_PROPS

    return ( true );
}
```

```
bool Update( void )
{
    static int          count = 0;

    //
    // Don't perform any calculations, just mark the state as having
    // been modified.
    //

    quanta->modify();
#ifdef TWO_PROPS
    quanta2->modify();
#endif // end TWO_PROPS

    return ( true );
}


bool Destruct( void )
{
    return ( true );
}
```

Figure B.2 Benchmark entity source code.

```
#include <stdarg.h>
#include "Manager.h"


bool    RegisterInterest( int noofParams, ... );
bool    Construct( int noofParams, ... );
bool    Destruct( int noofParams, ... );
bool    Update( int noofParams, ... );


Manager         *manager;


int main( int argc, char *argv[] )
{
    if ( argc != 2 || argv[1][0] != '-' )
    {
        cerr << "usage: " << argv[0] << " [-qt]\n";
        return ( 1 );
    }

    Process::IPCipc = Process::IPC::IPC_NONE;

    if ( argv[1][1] == 'q' )
    {
        ipc |= Process::IPC::IPC_QNX;
    }
    if ( argv[1][1] == 't' )
    {
        ipc |= Process::IPC::IPC_TCPIP;
    }
```

```
    try
    {
        manager = new Manager( ipc, argv[0] );

        manager->callback( MSG_UML_INIT_DEF, RegisterInterest );
        manager->callback( MSG_UML_CONSTRUCT, Construct );
        manager->callback( MSG_UML_DESTRUCT, Destruct );
        manager->callback( MSG_UML_UPDATE, Update );

        manager->serviceEvents();
    }
    catch ( ManagerCTORError )
    {
        cerr << argv[0] << ": failed to construct... terminating\n";
        return ( 1 );
    }
    catch ( ManagerError )
    {
        delete manager;
        cerr << argv[0] << ": terminating\n";
        return ( 1 );
    }

    return ( 0 );
}


bool RegisterInterest( int noofParams, ... )
{
    //
    // Identify which elements of the UML description we will want
    // to monitor.
    //

    manager->monitor( "Benchmark.quanta" );
#ifdef TWO_PROPS
    manager->monitor( "Benchmark.quanta2" );
#endif // end TWO_PROPS

    return ( true );
}


bool Construct( int noofParams, ... )
{
    va_list     params;

    if ( noofParams != 2 )
    {
        cerr << "Construct: expecting 2 parameters for callback\n";
        return ( false );
    }

    va_start( params, noofParams );

    Manager::Monitor    *mon = va_arg( params, Manager::Monitor* );
    Manager::Entity     *ent = va_arg( params, Manager::Entity* );

    va_end( params );

    return ( true );
}
```

```
bool Destruct( int noofParams, ... )
{
    return ( true );
}


bool Update( int noofParams, ... )
{
    va_list     params;

    if ( noofParams != 2 )
    {
        cerr << "Update: expecting 2 parameters for callback\n";
        return ( false );
    }

    va_start( params, noofParams );

    Manager::Monitor    *mon = va_arg( params, Manager::Monitor* );
    Manager::Entity     *ent = va_arg( params, Manager::Entity* );

    va_end( params );

    return ( true );
}
```

Figure B.3 Benchmark manager source code.

Appendix C

UML Benchmark Results

The charts for this appendix and UML source code can be downloaded via anonymous ftp from:

ftp://ftp.dcs.ed.ac.uk/pub/rjh/uml

Appendix D

PML Benchmark Results

The charts for this appendix can be downloaded via anonymous ftp from:

ftp://ftp.dcs.ed.ac.uk/pub/rjh/pml

Appendix E

UM Benchmark Results

The charts for this appendix can be downloaded via anonymous ftp from:

ftp://ftp.dcs.ed.ac.uk/pub/rjh/um