OSEK / VDX

System Generation

OIL: OSEK Implementation Language

Version 2.3

September 10th, 2001

Preface

OSEK/VDX is a joint project of the automotive industry. It aims at an industry standard for an open-ended architecture for distributed control units in vehicles.

For detailed information about OSEK project goals and partners, please refer to the “OSEK Binding Specification”.

This document describes the OSEK implementation language (OIL) concept of the description for the OSEK real-time operating system, capable of multitasking, which can be used for motor vehicles. It is not a product description which relates to a specific implementation.

General conventions, explanations of terms and abbreviations have been compiled in the additional inter-project "OSEK Overall Glossary".

Contents

List of Figures

1 Introduction

1.1 General Remarks

This document refers to the OSEK OS specification 2.2 of the operating system. For a better understanding of the document the reader should be familiar with the contents of the OS specification.

1.2 Motivation

Today within the OSEK OS standardisation, only the runtime services and API are defined. To reach the original goal of OSEK of portable software, a way has been defined to describe the configuration of an OSEK application and operating system .

This description version 2.3 of an OSEK system only addresses a **single** central processing unit (CPU) in an electronic control unit (ECU), not an ECU network.

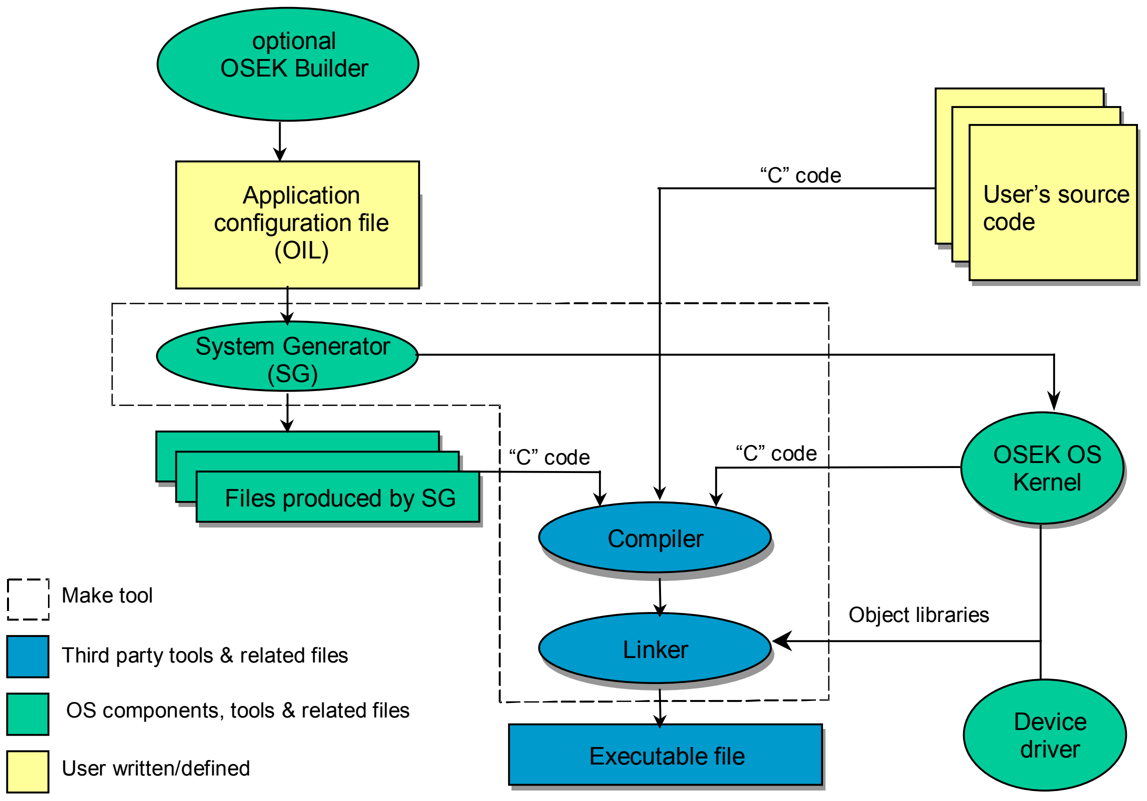

Figure 1-1: Example of development process for OSEK/VDX applications

The figure above shows an example of a development process for OSEK/VDX applications. The OIL description may be hand-written or generated by a system configuration tool. Operating systems delivered in source code are compiled together with the application, others delivered as a library are integrated by the linker.

1.3 Acronyms

API	Application Program Interface
BNF	Backus-Naur Form[1]
COM	Communication
CPU	Central Processing Unit
ECU	Electronic Control Unit
ISR	Interrupt Service Routine
NM	Network Management
OIL	OSEK Implementation Language
OS	Operating System
UML	Unified Modeling Language

[1] NAUR, Peter (ed.), "Revised Report on the Algorithmic Language ALGOL 60.", Communications of the ACM, Vol. 3, No.5, pp. 299-314, May 1960 or
M. Marcotty & H. Ledgard, The World of Programming Languages, Springer-Verlag, Berlin 1986., pages 41 and following.

2 Language Definition

2.1 Preamble

The goal of OIL is to provide a possibility to configure an OSEK application inside a particular CPU. This means for each CPU there is one OIL description.

All OSEK system objects are described using the OIL format.

2.2 General Concept

The OIL description of the OSEK application is considered to be composed of a set of system objects. A CPU is a container for these application system objects.

OIL defines standard types for system objects. Each object is described by a set of attributes and references. OIL defines explicitly all **standard attributes** for each standard system object.

The possible references defined in OIL for standard objects are presented in the following picture.

Each OSEK implementation can define additional specific attributes and references. Only attributes may be added to the existing objects. Creating of new objects or other changes to the grammar are not allowed. All non-standard attributes (**optional attributes**) are considered to be fully implementation specific and have no standard interpretation. Each OSEK implementation can limit the given set of values for object attributes (e.g. restrict the possible value range for priorities).

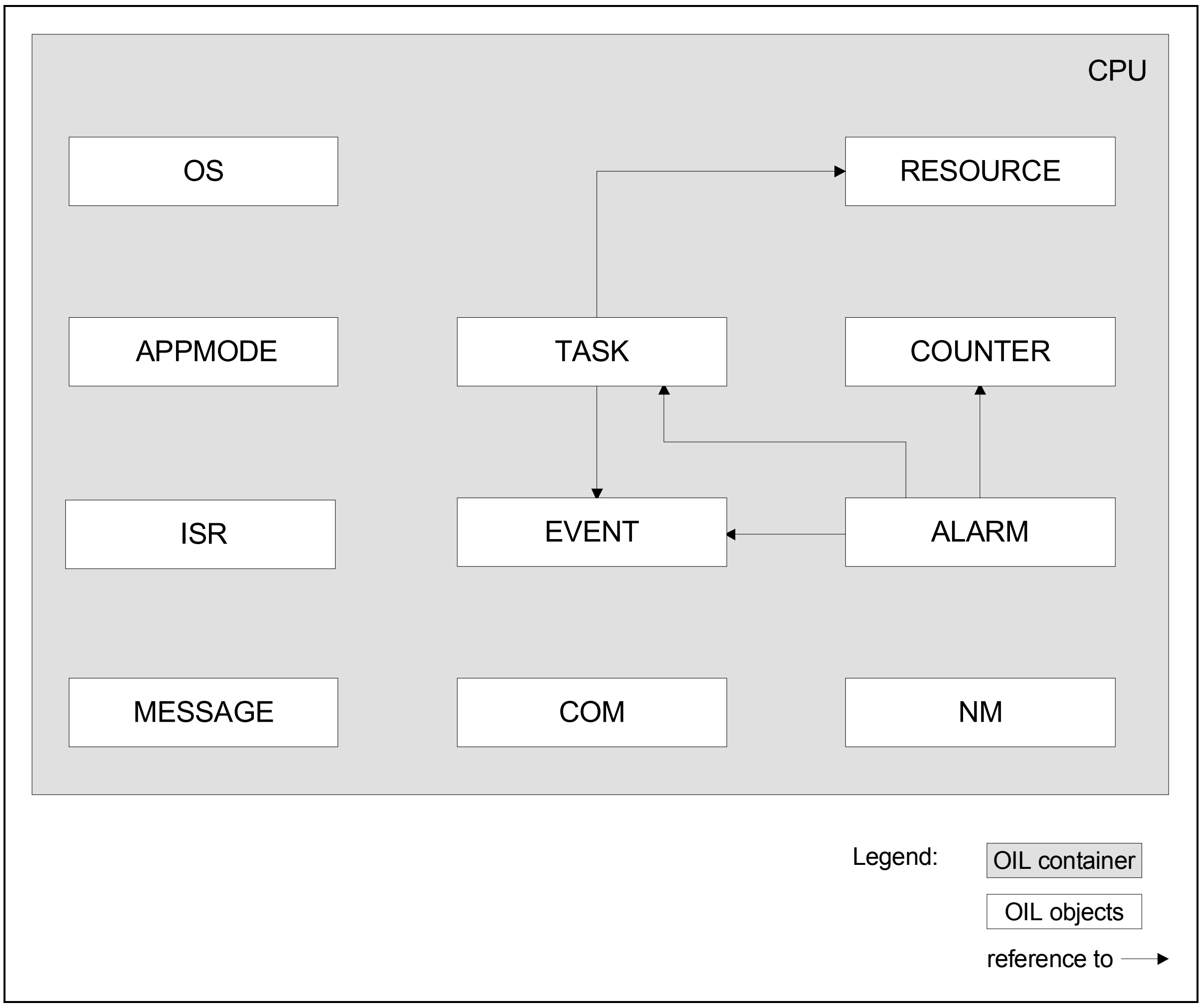

Figure 2-1: OIL standard objects

Description of the OIL objects:

CPU:	the CPU on which the application runs under the OSEK OS control.
OS:	the OSEK OS that runs on the CPU. No standard references are defined in OIL from OS to other system objects but, of course, all system objects are controlled by OS.
APPMODE:	defines different modes of operation for the application. No standard attributes for the APPMODE object.
ISR:	interrupt service routines supported by OS.
RESOURCE:	the resource that can be occupied by a task.
TASK:	the task handled by the OS.
COUNTER:	the counter represents hardware/software tick source for alarms.
EVENT:	the event tasks may react on.
ALARM:	the alarm is based on a counter and can either activate a task or set an event or activate an alarm-callback routine.

MESSAGE:	the message is defined in OSEK COM and defines a mechanism for data exchange between different entities (entities beeing tasks or ISRs).
COM:	the communication subsystem. The COM object has standard attributes to define general properties for the COM.
NM:	the network management subsystem. This subsystem is out of the scope of the OSEK OS specification, so no standard attributes and references are defined for the NM object.

2.3 OIL Basics

2.3.1 OIL file structure

The OIL description contains two parts - one for the definition of standard and implementation specific features (implementation definition) and another one for the definition of the structure of the application located on the particular CPU (application definition).

The OIL description consists of one main OIL-file which can reference to includes (see section 2.3.9).

2.3.2 Syntax

The grammar rules for an OIL file are presented in the document using a notation similar to the Backus-Naur Form (BNF), see section 5.2, Syntax of OIL.

All keywords, attributes, object names, and other identifiers are case-sensitive.

Comments in the BNF notation are written as C^{++}-style comments.

2.3.3 OIL versions

OIL version "2.0" corresponds to the OSEK OS specification 2.0 revision 1.

OIL version "2.1" also corresponds to the OSEK OS specification 2.0 revision 1. OIL version "2.1" means an OIL-internal extension in syntax and semantics.

The OIL version “2.1” is not compatible to the OIL version “2.0”.

The OIL version “2.2” is compatible to OIL “2.1”. “2.2” only defines new standard attributes.

The OIL version “2.3” corresponds to the OS specification 2.2 and is compatible to OIL “2.2”. “2.3” only defines new standard attributes.

2.3.4 Implementation Definition

For each object, the implementation definition defines all attributes and their properties for a particular OSEK implementation.

The implementation definition must be present in the OIL description and must contain all standard attributes, which are listed in chapter 3.2. The value range of those attributes may be restricted. Attribute definition is described in chapter 4.

Additional attributes and their properties can be defined for the objects for the particular OSEK implementation. Additional attributes are optional.

The include mechanism (see chapter 2.3.1, OIL file) can be used to define the implementation definition as a separate file. Thus corresponding implementation definition files can be developed and delivered with particular OSEK implementations and then included with the application definition in user's OIL files.

2.3.5 Application Definition

The application definition comprises a set of system objects and the values for their attributes. Except for the OS object the application definition can contain more than one system object of a particular type.

Each object is characterized by a set of attributes and their values. No attribute may appear that is not defined in the implementation definition. Attribute values must comply with the attribute properties specified in the implementation definition.

Attributes that take a single value may only be specified once per object. Attributes that take a list of values have to be specified as multiple statements.

Example for multiple statement
RESOURCE = RES1;
RESOURCE = RES2;

2.3.6 Dependencies between attributes

The OIL Specification allows to express dependencies between attributes. To be more open to vendor specific and standard extensions the OIL syntax covers conditional attributes (parameters).

OIL allows infinite nesting of those dependencies.

Only ENUM and BOOLEAN attribute values can be parameterized.

If attributes in several sets of one conditional attribute have the same name they must have the same type.

Example (implementation and application part of OIL file)

```
/* implementation part */

MESSAGE {
   ENUM [
      ACTIVATETASK { TASK_TYPE Task; },
      SETEVENT { TASK_TYPE Task; EVENT_TYPE EventToSet; },

      NONE
   ] ACTION[];
};


/* application part */

MESSAGE ProdMessage {
  ACTION = SETEVENT {
     Task = TaskProd:"Producer Task";
     EventToSet = MessageEvent:"Producer Task informed by MessageEvent";
  };

  ACTION = ACTIVATETASK {
     Task = TaskToNotify:"Notified Task";
  };
};
```

2.3.7 Automatic attribute assignment

Attribute values may be calculated by the generator. For those attributes the keyword WITH_AUTO has to be defined in the implementation definition. In conjunction with WITH_AUTO the attribute value AUTO is valid in the application part and as a default value.

2.3.8 Default values

Default values are used by the generator in the case that an attribute is missing in the application definition.

Default values are mandatory for optional attributes. Because the syntax of the implementation specific part requires the definition of default values a special default value NO_DEFAULT is defined to explicitly suppress the default mechanism. In that case the attribute must be defined in the application part.

Default values are forbidden for standard attributes.

It is an error if a standard attribute is missing from the application definition.

The OIL grammar uses assignment in the implementation definition to specify default values.

List of all possible combinations of attributes with default values are shown in the example of ENUM.

The OIL syntax allows six combinations for the implementation-specific part and three combinations for the application part:

Implementation part	Application part		
	`param = A;`	`param = AUTO;`	`// nothing`
`ENUM [A, B, C] param = B;`	param ⇨A	ERROR	Param ⇨B
`ENUM [A, B, C] param = NO_DEFAULT;`	param ⇨A	ERROR	ERROR
`ENUM [A, B, C] param = AUTO;`	ERROR	ERROR	ERROR
`ENUM` **`WITH_AUTO`** `[A, B, C] param = B;`	param ⇨A	Generator-specific	param ⇨B
`ENUM` **`WITH_AUTO`** `[A, B, C] param = NO_DEFAULT;`	param ⇨A	Generator-specific	ERROR
`ENUM` **`WITH_AUTO`** `[A, B, C] param = AUTO;`	param ⇨A	Generator-specific	Generator-specific

Example:
```
IMPLEMENTATION myOS {
   TASK {
   UINT32 [1..0xff] STACKSIZE = 16; // If STACKSIZE is missing,
                                    // 16 is used as a default
   };
};
```

2.3.9 Include-mechanism

The include mechanism allows for separate definitions for some parts of OIL. The implementation definition can be delivered with an OSEK implementation and used (included) by the system designer.

The include statement has the same syntax as in ISO/ANSI-C:

```
#include <file>
#include "file"
```

- For each OIL tool there must be a way to specify search-paths for include files.
- `#include <file>` uses the search-path
- `#include "file"` uses the directory, where the including file resides

Placement of include directives

The same rules apply as for ISO/ANSI-C, e.g. the include statement has to be set on a seperate line and can be set anywhere in the description files.

2.3.10 Comments

The OIL file may contain C^{++}-style comments (`/* */` and `//`). C^{++} rules apply.

2.3.11 Descriptions

To describe OIL objects, attributes, and values, the OIL syntax offers the concept of descriptions. Descriptions are optional. They start after a colon (:), are enclosed in double quotes ("), and must not contain a double quote.

Example:

```
...
BOOLEAN START = FALSE:"Automatic start of alarm on system start";
...
```

Descriptions give the user additional information about OIL objects, attributes and values in a well defined format. The interpretation of descriptions is implementation specific.

3 Standard System Object Definitions

3.1 Rules

The application configuration files must conform some rules to be successfully processed. Those rules are:

- All objects are described using the OIL syntax.
- Each object must have a unique name. Each object may be divided into several parts.
- All object names must be accessible from the application.
- An attribute defines some object properties (for example, the task priority). Attributes that take a single value may only be specified once per object. Attributes that take a list of values must be specified as multiple statements.
- An object can have a set of references to other system objects. Per object there may be more than one reference to the same type of object (e.g. more than one reference to different events; see example in chapter 3.2.4.8).
- Values must be defined for all standard attributes of all objects, except for multiple attributes which can be empty.
- The `<name>` non-terminal represents any ISO/ANSI-C identifier.
- The `<number>` non-terminal represents any integer constant. The range of integers is according to the target platform. Notations: decimal and hexadecimal. Decimal integers with leading zeroes are not allowed as they might be misinterpreted as octal values!
- The `<string>` non-terminal represents any 8-bit character sequence enclosed in double-quotes ("), but not containing double-quotes.
- The description represents any 8-bit character sequence enclosed in double-quotes ("), but not containing double-quotes.
- A reference defines a uni-directional link to another object (for example, the task X has to be activated when the alarm Y expires).
- Implementation-specific additional parameters are only allowed for optional attributes. For standard attributes it is forbidden to define implementation-specific additional parameters[2].

[2] for portability reasons

3.2 Standard system objects, attributes, and references

For each system object the standard set of attributes with possible values is defined. These standard object characteristics must be supported by any implementation.

3.2.1 CPU

CPU is used as a container for application objects.

3.2.2 OS[3]

OS is the system object used to define OSEK operating system properties for an OSEK application.

In a CPU exactly one OS object has to be defined.

3.2.2.1 STATUS

The STATUS attribute specifies whether a system with standard or extended status has to be used. Automatic assignment is not supported for this attribute.

This attribute is an ENUM with following possible values:
- STANDARD
- EXTENDED

3.2.2.2 Hook routines

The following attribute names are defined for the hook routines supported by the OS:

- STARTUPHOOK
- ERRORHOOK
- SHUTDOWNHOOK
- PRETASKHOOK
- POSTTASKHOOK

- These attributes are of type BOOLEAN.

If a hook routine will be used, the value is set to TRUE otherwise the value is set to FALSE.

The usage of the access macros to the service ID and the context related information in the error hook is enabled by the following attributes of type BOOLEAN:

- USEGETSERVICEID
- USEPARAMETERACCESS

3.2.2.3 Sample

```
OS sampleOS {
   STATUS = STANDARD;
   STARTUPHOOK = TRUE;
   ERRORHOOK = TRUE;
   SHUTDOWNHOOK = TRUE;
```

[3] Attributes for Conformance Class and Scheduling are not defined as these are not part of the OS specification

```
   PRETASKHOOK = FALSE;
   POSTTASKHOOK = FALSE;
   USEGETSERVICEID = FALSE;
   USEPARAMETERACCESS = FALSE;
};
```

3.2.3 APPMODE

APPMODE is the system object used to define OSEK operating system properties for an OSEK OS application mode.

No standard attributes are defined for the APPMODE object.

In a CPU at least one APPMODE object has to be defined.

3.2.4 TASK

TASK objects represent OSEK tasks.

3.2.4.1 PRIORITY

The priority of the task is defined by the value of the PRIORITY attribute. This value has to be understood as a relative value; this means the values of PRIORITY show only the relative ordering of the tasks.

This attribute has the type UINT32.

OSEK defines the lowest priority as zero (0), a bigger value of the PRIORITY attribute corresponds to a higher priority (compare OSEK specification 2.1, ch. 4.5).

3.2.4.2 SCHEDULE

The SCHEDULE attribute defines the preemptability of the task.

This attribute is an ENUM with following possible values:
- NON
- FULL

The FULL value of this attribute corresponds to a preemptable task, the NON value to a non-preemptable task.

If the SCHEDULE attribute is set to NON no internal resources may be assigned to this task.

3.2.4.3 ACTIVATION

The ACTIVATION attribute defines the maximum number of queued activation requests for the task. A value equal to "1" means that at any time only single activation is permitted for this task (see OSEK OS specification).

This attribute has the type UINT32.

3.2.4.4 AUTOSTART

The AUTOSTART attribute determines whether the task is activated during the system start-up procedure or not for some specific application modes.

This attribute is a BOOLEAN .

If the task should be activated during the system start-up, the value is set to TRUE otherwise the value is set to FALSE. When set to TRUE, a list of application modes is defined in the

APPMODE sub-attribute of type APPMODE_TYPE which defines in which application modes the task is auto-started.

3.2.4.5 RESOURCE

The RESOURCE reference is used to define a list of resources accessed by the task.

This attribute is a multiple reference (see chapter 4.2, Reference Types) of type RESOURCE_TYPE.

3.2.4.6 EVENT

The EVENT reference is used to define a list of events the extended task may react on.

This attribute is a multiple reference (see chapter 4.2, Reference Types) of type EVENT_TYPE.

3.2.4.7 ACCESSOR

The ACCESSOR is used to define multiple references to sent or received messages. In addition the parameters WITHOUTCOPY and ACCESSNAME are defined.

This attribute is a parametrized ENUM with following possible values:

- SENT {MESSAGE_TYPE MESSAGE; BOOLEAN WITHOUTCOPY;
  STRING ACCESSNAME;}
- RECEIVED { MESSAGE_TYPE MESSAGE; BOOLEAN WITHOUTCOPY;
  STRING ACCESSNAME;}

ACCESSOR = SENT

The MESSAGE reference parameter defines the message to be sent by the task.

This parameter is a single reference (see chapter 4.2, Reference Types) of type MESSAGE_TYPE.

The WITHOUTCOPY parameter specifies if a local copy of the message is used.

This attribute is a BOOLEAN.

The ACCESSNAME parameter defines the reference which can be used by the application to access the message data.

This attribute is a STRING.

ACCESSOR = RECEIVED

The MESSAGE reference parameter defines the message to be received by the task.

This parameter is a single reference (see chapter 4.2, Reference Types) of type MESSAGE_TYPE.

The WITHOUTCOPY parameter specifies if a local copy of the message is used.

This attribute is a BOOLEAN.

The ACCESSNAME parameter defines the reference which can be used by the application to access the message data.

This attribute is a STRING.

3.2.4.8 Sample

```
TASK TaskA {
   PRIORITY = 2;
   SCHEDULE = NON;
   ACTIVATION = 1;
   AUTOSTART = TRUE {
      APPMODE = AppMode1;
      APPMODE = AppMode2;
   }
   RESOURCE = resource1;
   RESOURCE = resource2;
   RESOURCE = resource3;
   EVENT = event1;
   EVENT = event2;
   ACCESSOR = SENT {
      MESSAGE= anyMessage1;
      WITHOUTCOPY= TRUE;
      ACCESSNAME= "anyMessage1Buffer";
   };
};
```

3.2.5 COUNTER

A COUNTER serves as a base for the ALARM mechanism.

3.2.5.1 MAXALLOWEDVALUE

The MAXALLOWEDVALUE attribute defines the maximum allowed counter value.

This attribute has the type UINT32.

3.2.5.2 TICKSPERBASE

The TICKSPERBASE attribute specifies the number of ticks required to reach a counter-specific unit. The interpretation is implementation specific.

This attribute has the type UINT32.

3.2.5.3 MINCYCLE

The MINCYCLE attribute specifies the minimum allowed number of counter ticks for a cyclic alarm linked to the counter.

This attribute has the type UINT32.

3.2.5.4 Sample

```
COUNTER Timer {
   MINCYCLE = 16;
   MAXALLOWEDVALUE = 127;
   TICKSPERBASE = 90;
};
```

3.2.6 ALARM

An ALARM may be used to asynchronously inform or activate a specific task. It is possible to start ALARMS automatically at system start-up depending on the application mode.

3.2.6.1 COUNTER

The COUNTER reference defines the counter assigned to this alarm. Only one counter has to be assigned to the alarm. Any alarm has to be assigned to a particular counter.

This attribute is a single reference (see chapter 4.2, Reference Types).

3.2.6.2 ACTION

The ACTION attribute defines which type of task notification is used when the alarm expires.

This attribute is a parametrized ENUM with following possible values:
- ACTIVATETASK {TASK_TYPE TASK;}
- SETEVENT {TASK_TYPE TASK; EVENT_TYPE EVENT;}
- ALARMCALLBACK {STRING ALARMCALLBACKNAME;}

For an alarm, only one action is allowed.

ACTION = ACTIVATETASK

The TASK reference parameter defines the task to be activated when the alarm expires.

This parameter is a single reference (see chapter 4.2, Reference Types) of type TASK_TYPE.

ACTION = SETEVENT

The TASK reference parameter defines the task for which the event is to be set. The EVENT reference parameter defines the event to be set when the alarm expires.

TASK is a single reference of type TASK_TYPE. EVENT is a single reference of type EVENT_TYPE.

ACTION = ALARMCALLBACK

The ALARMCALLBACKNAME parameter defines the name of the callback routine which is called when the alarm expires.

3.2.6.3 AUTOSTART

The AUTOSTART attribute of type BOOLEAN defines if an alarm is started automatically at system start-up depending on the application mode.

When set to TRUE sub-attributes are used to define the ALARMTIME, i.e. the time when the ALARM shall expire first, the CYCLETIME, i.e. the cycle time of a cyclic ALARM and a list of application modes (APPMODE) for which the AUTOSTART shall be performed.

```
BOOLEAN [
   TRUE {
      UINT32 ALARMTIME;
      UINT32 CYCLETIME;
      APPMODE_TYPE APPMODE[];
         },
   FALSE
] AUTOSTART;
```

3.2.6.4 Samples

```
ALARM WakeTaskA {
   COUNTER = Timer;
   ACTION = SETEVENT {
      TASK = TaskA;
      EVENT = event1;
   };
   AUTOSTART = FALSE;
};

ALARM WakeTaskB {
   COUNTER = SysCounter;
   ACTION = ACTIVATETASK {
      TASK = TaskB;
   };
   AUTOSTART = TRUE {
      ALARMTIME = 50;
      CYCLETIME = 100;
      APPMODE = AppMode1;
      APPMODE = AppMode2;
   }
};


ALARM ActivateCallbackC {
   COUNTER = SysCounter;
   ACTION = ALARMCALLBACK {
      ALARMCALLBACKNAME = "CallbackC";
   };
   AUTOSTART = FALSE;
};
```

3.2.7 RESOURCE

Resource is used to co-ordinate concurrent access of several tasks to a shared resource, e.g. the scheduler, any program sequence, memory or any hardware area.
There is one attribute of type ENUM defined to specify the RESOURCEPROPERTY. This attribute can take the following values:

- STANDARD: A normal resource which is not linked to another resource and is not an internal resource.
- LINKED: A resource which is linked to another resource with the property STANDARD or LINKED. The resource to which the linking shall be performed is defined by the sub-attribute LINKEDRESOURCE of type RESOURCE_TYPE. The code generator of the operating system must resolve chains of linked resources.
- INTERNAL: An internal resource which cannot be accessed by the application.

3.2.7.1 Sample

```
RESOURCE MsgAccess
{
   RESOURCEPROPERTY = STANDARD;
};
```

3.2.8 EVENT

An EVENT object is represented by its mask. The name of the event is a synonym for its mask.

The same event may be set for different tasks. Events with the same name are identical, therefore the event mask is identical. Events with the same mask are generally not identical i.e. their names may be different.

3.2.8.1 MASK

The event mask is an integer number MASK of type UINT64. The other way to assign an event mask is to declare it as AUTO. In this case, one bit is automatically assigned to the event mask. This bit is **unique** with respect to the tasks that reference the event.

3.2.8.2 Samples

```
EVENT event1 {
   MASK = 0x01;
};

EVENT event2 {
   MASK = AUTO;
};
```

In the C-Code the user is allowed to combine normal event masks and AUTO event masks.

```
C-Code:
   ...
   WaitEvent ( event1 | event2 );
   ...
```

Example for the same event object (same event name) for events used by different tasks:
```
EVENT emergency {
   MASK = AUTO;
};

TASK task1 {
   EVENT = myEvent1;
   EVENT = emergency;
};

TASK task2 {
   EVENT = emergency;
   EVENT = myEvent2;
};

TASK task7 {
   EVENT = emergency;
   EVENT = myEvent2;
};
```

In the C-Code the user is allowed to use the emergency event with all three tasks.
```
C-Code:
   ...
   SetEvent (task1, emergency);
   SetEvent (task2, emergency);
   SetEvent (task7, emergency);
   ...
```

Another use for the same event name for events of different tasks is in control loops:
```
C-Code:
...
TaskType myList[] = {task1, task2, task7};
int myListLen = 3;
int i=0;
for (i=0;i<myListLen;i++) {
   SetEvent(myList[i],emergency);
}
...
```

3.2.9 ISR

ISR objects represent OSEK interrupt service routines (ISR).

3.2.9.1 CATEGORY

The CATEGORY attribute defines the category of the ISR. This attribute is a UINT32, only values of 1and 2 are allowed.

3.2.9.2 RESOURCE

The RESOURCE reference is used to define a list of resources accessed by the ISR.

This attribute is a multiple reference (see chapter 4.2, Reference Types) of type RESOURCE_TYPE.

3.2.9.3 ACCESSOR

The ACCESSOR is used to define multiple references to sent or received messages. In addition the parameter ACCESSNAME is defined.

This attribute is a parametrized ENUM with following possible values:

- SENT {MESSAGE_TYPE MESSAGE; STRING ACCESSNAME;}
- RECEIVED { MESSAGE_TYPE MESSAGE; STRING ACCESSNAME;}

ACCESSOR = SENT

The MESSAGE reference parameter defines the message to be sent by the ISR.

This parameter is a single reference (see chapter 4.2, Reference Types) of type MESSAGE_TYPE.

The ACCESSNAME parameter defines the reference which can be used by the application to access the message data.

This attribute is a STRING.

ACCESSOR = RECEIVED

The MESSAGE reference parameter defines the message to be received by the ISR.

This parameter is a single reference (see chapter 4.2, Reference Types) of type MESSAGE_TYPE.

The ACCESSNAME parameter defines the reference which can be used by the application to access the message data.

This attribute is a STRING.

3.2.9.4 Sample

```
ISR TimerInterrupt {
   CATEGORY = 2;
   RESOURCE = someResource;
   ACCESSOR = RECEIVED {
      MESSAGE= anyMessage2;
      ACCESSNAME= "anyMessage2Buffer";
   };
};
```

3.2.10 MESSAGE

MESSAGE objects represent OSEK messages.

3.2.10.1 TYPE

The TYPE attribute defines if the message has a queue or not.

This attribute is a parametrized ENUM with following possible values:

- UNQUEUED {}
- QUEUED {UINT64 QUEUEDEPTH;}

TYPE = UNQUEUED

No queue is available for the message. No subattributes are defined.

TYPE = QUEUED

A queue of QUEUEDEPTH elements is defined.

QUEUEDEPTH is of type UINT64.

The QUEUED attributevalue specifies if the message has a queue. If used for internal communication the COM conformance class will be CCCB.

3.2.10.2 CDATATYPE

The CDATATYPE parameter describes the data type of message data according to the C-language (e.g. *int* or a structure name).

This attribute is a STRING.

3.2.10.3 ACTION

The ACTION attribute defines which type of task notification is used when the message is received. For unqueued messages more than one action per message is possible.

This attribute is a parametrized ENUM with following possible values:

- NONE {}
- ACTIVATETASK {TASK_TYPE TASK;}
- SETEVENT {TASK_TYPE TASK; EVENT_TYPE EVENT;}
- CALLBACK {STRING CALLBACKNAME;}
- FLAG {STRING FLAGNAME;}

For unqueued messages a list of actions is allowed according to OSEK COM specification 2.2.

For queued messages one action is allowed.

ACTION = NONE

No action is performed by sending this message.

ACTION = ACTIVATETASK

The TASK reference parameter defines the task to be activated when the message is sent.

This parameter is a single reference (see chapter 4.2, Reference Types) of type TASK_TYPE.

ACTION = SETEVENT

The TASK reference parameter defines the task for which the event is to be set. The EVENT reference parameter defines the event to be set when the message is sent.

TASK is a single reference of type TASK_TYPE. EVENT is a single reference of type EVENT_TYPE.

ACTION = CALLBACK

The CALLBACK parameter defines the function which is called when the message is sent.

This attribute is a STRING.

ACTION = FLAG

The FLAG parameter defines the name of the flag which is set when the message is sent.

This attribute is a STRING.

If used for internal communication the COM conformance class will be CCCB.

3.2.10.4 Sample

```
MESSAGE newInfos   {
   TYPE = UNQUEUED;
   CDATATYPE = "infoStruct";
   ACTION= SETEVENT {
      TASK= anyTask;
      EVENT= anyEvent;
   };
   ACTION= SETEVENT {
      TASK= otherTask;
      EVENT= otherEvent;
   };
   ACTION= CALLBACK {
       CALLBACKNAME= "msgReceived1";
   };
};
```

3.2.11 COM

COM is the system object used to define OSEK COM communication subsystem properties.

In a CPU not more than one COM object can be defined.

3.2.11.1 USEMESSAGERESOURCE

The USEMESSAGERESOURCE attribute specifies if the message resource mechanism is used. If used for internal communication the COM conformance class will be CCCB.

This attribute has the type BOOLEAN.

3.2.11.2 USEMESSAGESTATUS

The USEMESSAGESTATUS attribute specifies if the message status is available. If used for internal communication the COM conformance class will be CCCB.

This attribute has the type BOOLEAN.

3.2.11.3 Sample

```
COM sampleCOM {
   USEMESSAGERESOURCE = TRUE;
   USEMESSAGESTATUS = FALSE;
};
```

3.2.12 NM

NM objects represent the network management subsystems. No standard attributes are defined for the NM object.

4 Definition of Particular Implementation

OIL is intended to be used for the description of applications in any OSEK implementation. The implementation definition describes a set of attributes for each system object and valid values for these attributes. All standard attributes must be defined in this part. For standard attributes this part can only limit the value range, but in no case extend the value range or change the value type. Optional attributes must specify a default value, AUTO (if defined WITH_AUTO), or NO_DEFAULT.

The reference to an object or set of objects can be also defined by this part.

4.1 Attribute types

Any implementation specific attribute has to be defined before it is used.

The attribute type and attribute value range (if it exists) has to be defined. The range of attribute values can be defined in two ways: either the minimum and maximum allowed attribute values are defined (the [0..12] style) or the list of possible attribute values are presented. A mix of both is not allowed.

The WITH_AUTO specifier can be combined with any attribute type except for references. If WITH_AUTO is specified the attribute can have the value AUTO and the possibility of automatic assignment by an off-line tool.

OIL data types are listed below. Note that these data types are not necessarily the same as the corresponding C data types.

4.1.1 UINT32

Any unsigned integer number (possibly restricted to a range of numbers, see <impl_attr_def> chapter 5.2, Syntax of OIL).

```
UINT32 [1..255] NON_SUSPENDED_TASKS;
UINT32 [0,2,3,5] FreeInterrupts;
UINT32 aNumber;
```

This data type allows to express any 32 bit value in the range of [0..(2^{32}-1)].

4.1.2 INT32

Any signed integer number in the range of [-2^{31}..(2^{31}-1)].

4.1.3 UINT64

Any unsigned integer number in the range [0..(2^{64}-1)]

4.1.4 INT64

Any signed integer number in the range [-2^{63}..(2^{63}-1)].

4.1.5 FLOAT

Any floating point number according to IEEE-754 standard (Range: +/- 1,176E-38 to +/- 3,402E+38).

```
FLOAT [1.0 .. 25.3] ClockFrequency; // Clock frequency in Mhz
```

4.1.6 ENUM

ENUM defines a list of ISO/ANSI-C enumerators. Any enumerator from this list can be assigned to an attribute of the according type.

```
ENUM [NON, FULL] SCHEDULE;
ENUM [mon, tue, wed, thu, fri] myWeek;
```

ENUM types can be parameterized, i.e. the particular enumerators can have parameters. The parameter specification is denoted in curly braces after the enumerator. Any kind of attribute type is allowed as parameter of an enumerator.

```
ENUM [
   ACTIVATETASK {TASK_TYPE TASK;},
   SETEVENT {TASK_TYPE TASK; EVENT_TYPE EVENT;}
] ACTION;
```

4.1.7 BOOLEAN

The attribute of this type can have either TRUE or FALSE value.

```
BOOLEAN DontDoIt;
...
DontDoIt = FALSE;
```

BOOLEAN types can be parameterized, i.e. the particular boolean values can have parameters. Parameter specification are denoted in curly braces after an explicit enumeration of the boolean values. Any kind of attribute type is allowed as parameter of a boolean value.

```
BOOLEAN [
   TRUE {TASK_TYPE TASK; EVENT_TYPE EVENT;},
   FALSE {TASK_TYPE TASK;}
] IsEvent;
```

4.1.8 STRING

Any 8-bit character sequence enclosed in double-quotes, but not containing double-quotes, can be assigned to this attribute.

4.2 Reference Types

A reference type is a data type that refers to a system object, e.g. to a task, to an event, to an alarm, etc.

Reference types can be used to establish links between system objects, e.g. within an alarm description a reference type attribute can refer a task object that is to be activated by the alarm.

The definition of a reference type specifies which type of system objects are referred, e.g. the referred system objects are of type TASK, of type EVENT, of type ALARM, etc.

The reference type is taken from the referenced object (e.g. a reference to a task shall use the TASK_TYPE keyword as reference type). A reference can refer to any system object.

A single reference type refers to exactly one object.

A definition of a single reference type consists of the system object type to be referred followed by the symbolic name of the reference type being defined.

4.3 Multiple values

It is possible to use one attribute name to refer to a set of values of the same type. The set may be empty. For example, the EVENT attribute of a task object can refer to a set of events. Multiple values are allowed for all types.

A definition of a multiple reference type consists of the system object type to be referred followed by the symbolic name of the reference type being defined followed by an empty pair of brackets '[]'.

Example: `EVENT_TYPE MYEVENTS[];`

A definition of a multiple attribute is the symbolic name followed by an empty pair of brackets '[]'.

Example: `INT32 InterruptNumber[];`

A definition of a multiple parameterized ENUM or BOOLEAN type consists of the ENUM (or BOOLEAN) type definition followed by an empty pair of brackets '[]'.

```
IMPLEMENTATION x {
   ...
   MESSAGE {
      ENUM [
         ACTIVATETASK
         {
            TASK_TYPE TASK;
         }: "Task to be activated",
         SETEVENT
         {
            TASK_TYPE TASK;
            EVENT_TYPE EVENT;
         }: "Event to be set",
         NONE
      ] ACTION[];
   };
};
CPU y {
   ...
   MESSAGE newInfos // broadcast
   {
      ITEMTYPE = "infoStruct";
      ITEMS = 1;
      ACTION = ACTIVATETASK {TASK= taskA;};
      ACTION = ACTIVATETASK {TASK= taskB;};
   };
};
```

4.4 Sample

The implementation can define some additional attributes for a standard object or restrict the value range of standard attributes.

The example below shows:

1. The limitation of the ENUM value range for the standard OS attribute STATUS.
2. The definition of an implementation specific attribute NON_SUSPENDED_TASKS of type UINT32 with a value range.
3. The limitation of the UINT32 value range for the standard task attribute PRIORITY.
4. The default value for StackSize is set to 16.
5. The limitation of the ENUM value range for the standard alarm attribute ACTION.
6. The definition of an implementation specific attribute START of type BOOLEAN for alarms.
7. The definition of an implementation specific attribute ITEMTYPE of type STRING for messages.
8. The definition of a reference to message objects for ISRs.
9. The possible usage of the defined or modified attributes in the application definition.
10. Separation of the object MyTask1 into two definitions.

```
IMPLEMENTATION SpecialOS {
   OS {

      ENUM [EXTENDED] STATUS;
      UINT32 [1..255] NON_SUSPENDED_TASKS = 16;
      ...
   };

   TASK {
      UINT32 [1 .. 256] PRIORITY;    // define range of standard
                                     // attribute PRIORITY

      INT32 StackSize= 16;           // stacksize in bytes for a task
      ...
   };


   ALARM {
      ENUM [ACTIVATETASK {TASK_TYPE TASK;}] ACTION;
      // define possible value(s) of standard attribute ACTION
      BOOLEAN START = FALSE;   // define implementation specific
                               // attribute START of type BOOLEAN
      ...
   };


   MESSAGE {
      STRING ITEMTYPE = "";          // define implementation specific
                                     // attribute ITEMTYPE of type STRING
      ...
};
```

```
   ISR {
      MESSAGE_TYPE RCV_MESSAGES[] = NO_DEFAULT;
                                     // define implementation specific
                                     // attribute RCV_MESSAGES of type
                                     // 'multiple reference to objects
                                     // of type MESSAGE'
      ...
   };


}; // End IMPLEMENTATION SpecialOS


CPU SampleCPU {
   OS MyOs {

      ...

   };


   TASK MyTask1 {
      PRIORITY = 17;
      ...
   };
   TASK MyTask1 {
      StackSize = 64;
      ...
   };
   ALARM MyAlarm1 {
      ACTION = ACTIVATETASK {
         TASK = MyTask1;
      };
      START = TRUE;
      ...
   };
   MESSAGE MyMsg1 {
      ITEMTYPE = "SensorData";
      ...
   };
   MESSAGE MyMsg2 {
      ITEMTYPE = "Acknowledge";
      ...
   };
   ISR MyIsr1 {
      RCV_MESSAGES = MyMsg1;
      RCV_MESSAGES = MyMsg2;
      ...
   };
}; // End CPU SampleCPU
```

This example is not a complete OIL file therefore the dots represent missing parts.

5 Appendix

5.1 Static model of OIL

The following figure shows the static model of OIL in UML (unified modeling language) notation.

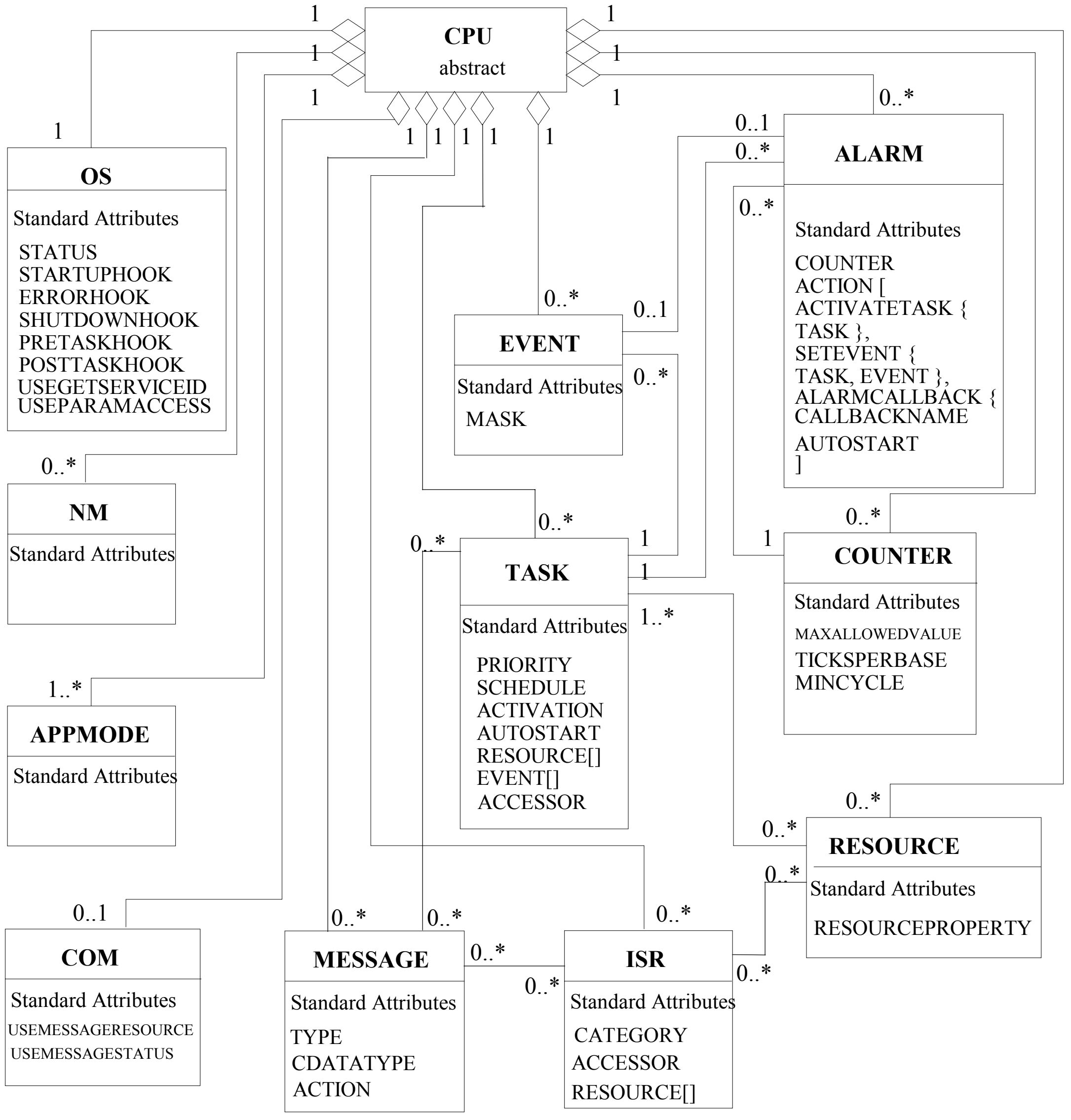

Figure 5-1: Static model of OIL[4]

[4] The UML-Diagram does not show the relations between ACTIONs of messages and TASK or EVENT

5.2 Syntax of OIL

The OIL file has the following structure:

```
<file> ::=
      <OIL_version>
      <implementation_definition>
      <application_definition>

<OIL_version> ::=
      "OIL_VERSION" "=" <version> <description> ";"

<version> ::= <string>

<implementation_definition> ::=
      "IMPLEMENTATION" <name> "{" <implementation_spec_list> "}"
      <description> ";"

<implementation_spec_list> ::=
      <implementation_spec>
      | <implementation_spec_list> <implementation_spec>

<implementation_spec> ::=
      <object> "{" <implementation_list> "}" <description> ";"

<object> ::=
      "OS" | "TASK" | "COUNTER" | "ALARM" | "RESOURCE" | "EVENT" | "ISR"
      | "MESSAGE" | "COM" | "NM" | "APPMODE"

<implementation_list> ::=
        /* empty list */
      | <implementation_def>
      | <implementation_list> <implementation_def>

<implementation_def> ::= <impl_attr_def> | <impl_ref_def>

<impl_attr_def> ::=
      "UINT32" <auto_specifier> <number_range> <attribute_name>
              <multiple_specifier><default_number> <description> ";"
      | "INT32" <auto_specifier> <number_range> <attribute_name>
             <multiple_specifier> <default_number> <description> ";"
      | "UINT64" <auto_specifier> <number_range> <attribute_name>
             <multiple_specifier> <default_number> <description> ";"
      | "INT64" <auto_specifier> <number_range> <attribute_name>
             <multiple_specifier> <default_number>  <description> ";"
      | "FLOAT" <auto_specifier> <float_range> <attribute_name>
             <multiple_specifier> <default_float>  <description> ";"
      | "ENUM" <auto_specifier> <enumeration> <attribute_name>
             <multiple_specifier> <default_name>  <description> ";"
      | "STRING" <auto_specifier> <attribute_name>
             <multiple_specifier> <default_string>  <description> ";"
      | "BOOLEAN" <auto_specifier> <bool_values> <attribute_name>
             <multiple_specifier> <default_bool>  <description> ";"

<impl_parameter_list> ::=
      /* empty definition */
      |"{" <impl_def_list> "}"

<impl_def_list> ::=
      /* empty definition */
      | <implementation_def>
      | <implementation_def> <impl_def_list>
```

```
<auto_specifier> ::=
      /* empty definition */
      | "WITH_AUTO"

<number_range> ::=
      /* empty definition */
      | "[" <number> ".." <number> "]"
      | "[" <number_list> "]"

<number_list> ::=
      <number> | <number_list> "," <number>

<default_number> ::=
      /* empty definition */
      | "=" <number> | "=" "NO_DEFAULT" | "=" "AUTO"

<description> ::=
      /* empty definition */
      | ":" <string>

<float_range> ::=
      /* empty definition */
      | "[" <float> ".." <float> "]"

<default_float> ::=
      /* empty definition */
      | "=" <float> | "=" "NO_DEFAULT" | "=" "AUTO"

<enumeration> ::=
      "[" <enumerator_list> "]"

<enumerator_list> ::=
      <enumerator>
      | <enumerator_list> "," <enumerator>

<enumerator> ::=
        <name> <description>
      | <name> <impl_parameter_list> <description>

<bool_values> ::=
      /* empty definition */
      | "[" "TRUE" <impl_parameter_list> <description> ","
              "FALSE" <impl_parameter_list> <description> "]"

<default_name> ::=
      /* empty definition */
      | "=" <name> | "=" "NO_DEFAULT" | "=" "AUTO"

<default_string> ::=
      /* empty definition */
      | "=" <string> | "=" "NO_DEFAULT" | "=" "AUTO"

<default_bool> ::=
      /* empty definition */
      | "=" <boolean> | "=" "NO_DEFAULT" | "=" "AUTO"

<impl_ref_def> ::=
      <object_ref_type> <reference_name> <multiple_specifier> <description>
      ";"
```

```
<object_ref_type> ::=
      "OS_TYPE" | "TASK_TYPE" | "COUNTER_TYPE" | "ALARM_TYPE"
      | "RESOURCE_TYPE" | "EVENT_TYPE" | "ISR_TYPE"
      | "MESSAGE_TYPE" | "COM_TYPE" | "NM_TYPE" | "APPMODE_TYPE"

<reference_name> ::= <name> | <object>

<multiple_specifier> ::=
      /* empty definition */
      | "[" "]"

<application_definition> ::=
      "CPU" <name> "{" <object_definition_list> "}" <description> ";"

<object_definition_list> ::=
      /* empty definition */
      | <object_definition>
      | <object_definition_list> <object_definition>

<object_definition> ::=
      <object_name> <description> ";"
      | <object_name> "{" <parameter_list> "}" <description> ";"

<object_name> ::= <object> <name>

<parameter_list> ::=
      /* empty definition */
      | <parameter>
      | <parameter_list> <parameter>

<parameter> ::=
      <attribute_name> "=" <attribute_value> <description> ";"

<attribute_name> ::= <name> | <object>


<attribute_value> ::=
      <name>
      | <name> "{" <parameter_list> "}"
      | <boolean>
      | <boolean> "{" <parameter_list> "}"
      | <number>
      | <float>
      | <string>
      | "AUTO"

<name> ::= Name

<string> ::= String
<boolean> ::= "FALSE" | "TRUE"

<number> ::= <dec_number> | <hex_number>

<dec_number>  ::=
      <sign> <int_digits>

<sign> ::=
      /* empty definition */
      | "+"
      | "-"
```

```
<int_digits> ::=
      <zero_digit>
      | <pos_digit>
      | <pos_digit> <dec_digits>

<dec_digits> ::=
      | <dec_digit>
      | <dec_digit> <dec_digits>

<float> ::=
      <sign> <dec_digits> "." <dec_digits> <exponent>

<exponent> ::=
      /* empty definition */
      | "e" <sign> <dec_digits>
      | "E" <sign> <dec_digits>

<zero_digit> ::=
      "0"

<pos_digit> ::=
      "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<dec_digit> ::= <zero_digit> | <pos_digit>

<hex_number> ::= "0x" <hex_digits>

<hex_digits> ::=
      <hex_digit>
      | <hex_digit> <hex_digits>

<hex_digit> ::=
      "A" | "B" | "C" | "D" | "E" | "F"
      | "a" | "b" | "c" | "d" | "e" | "f"
      | "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

5.3 Default definition of standard object attributes and references

The definition of standard attribute types and parameters can be presented in the following form[5]:

```
IMPLEMENTATION Standard {
      OS {
            ENUM [STANDARD, EXTENDED] STATUS;
            BOOLEAN STARTUPHOOK;
            BOOLEAN ERRORHOOK;
            BOOLEAN SHUTDOWNHOOK;
            BOOLEAN PRETASKHOOK;
            BOOLEAN POSTTASKHOOK;
            BOOLEAN USEGETSERVICEID;
            BOOLEAN USEPARAMETERACCESS;
      };
      APPMODE {
      };
      TASK {
            BOOLEAN [
               TRUE
               {
                   APPMODE_TYPE APPMODE[];
               },
               FALSE
            ] AUTOSTART;
            UINT32 PRIORITY;
            UINT32 ACTIVATION;
            ENUM [NON, FULL] SCHEDULE;
            EVENT_TYPE EVENT[];
            RESOURCE_TYPE RESOURCE[];
            ENUM [
                  SENT
                  {
                        MESSAGE_TYPE MESSAGE;
                        BOOLEAN WITHOUTCOPY;
                        STRING ACCESSNAME;
                  },
                  RECEIVED
                  {
                        MESSAGE_TYPE MESSAGE;
                        BOOLEAN WITHOUTCOPY;
                        STRING ACCESSNAME;
                  }
            ] ACCESSOR[];     };
      ISR {
            UINT32 [1, 2] CATEGORY;
            RESOURCE_TYPE RESOURCE[];
            ENUM [
                  SENT
                  {
                        MESSAGE_TYPE MESSAGE;
                        STRING ACCESSNAME;
                  },
                  RECEIVED
                  {
                        MESSAGE_TYPE MESSAGE;
                        STRING ACCESSNAME;
                  }
            ] ACCESSOR[];
      };
```

[5] Ordering of the elements is free.

```
COUNTER {
      UINT32 MINCYCLE;
      UINT32 MAXALLOWEDVALUE;
      UINT32 TICKSPERBASE;
};
ALARM {
      COUNTER_TYPE COUNTER;
      ENUM [
            ACTIVATETASK {TASK_TYPE TASK;},
            SETEVENT {TASK_TYPE TASK; EVENT_TYPE EVENT;}
            ALARMCALLBACK {STRING ALARMCALLBACKNAME;}
      ] ACTION;
      BOOLEAN [
         TRUE
         {
            UINT32 ALARMTIME;
            UINT32 CYCLETIME;
            APPMODE_TYPE APPMODE[];
         },
         FALSE
      ] AUTOSTART;

};
EVENT {
      UINT32 WITH_AUTO MASK;
};
RESOURCE {
      ENUM [
         STANDARD,
         LINKED {
            RESOURCE_TYPE LINKEDRESOURCE;
         },
         INTERNAL
      ] RESOURCEPROPERTY;

};
MESSAGE {
      ENUM [
            UNQUEUED,
            QUEUED {UINT64 QUEUEDEPTH;}
            ] TYPE;
      STRING CDATATYPE;
      ENUM [
            NONE,
            ACTIVATETASK
            {
                  TASK_TYPE TASK;
            },
            SETEVENT
            {
                  TASK_TYPE TASK;
                  EVENT_TYPE EVENT;
            },
            CALLBACK
            {
                  STRING CALLBACKNAME;
            },
            FLAG
            {
                  STRING FLAGNAME;
            }
      ] ACTION[]; // action to perform if message is sent or received
};
COM {
      BOOLEAN USEMESSAGERESOURCE;
```

```
            BOOLEAN USEMESSAGESTATUS;
            };
      NM {
      };
};
```

5.4 Sample of a complete OIL file

```
OIL_VERSION = "2.3";

IMPLEMENTATION MySpecificImplementation
{
   OS
   {
      ENUM [STANDARD, EXTENDED] STATUS;
      BOOLEAN STARTUPHOOK;
      BOOLEAN ERRORHOOK;
      BOOLEAN SHUTDOWNHOOK;
      BOOLEAN PRETASKHOOK;
      BOOLEAN POSTTASKHOOK;
      BOOLEAN USEGETSERVICEID;
      BOOLEAN USEPARAMETERACCESS;

      /* Implementation specific attribute added */
      UINT32 [1..255] NON_SUSPENDED_TASKS = 16;
      FLOAT [1.0 .. 25.3] ClockFrequency = 8.0; // Clock frequency in Mhz
   };

   APPMODE
   {
   };

   TASK
   {
      BOOLEAN [
          TRUE
          {
               APPMODE_TYPE APPMODE[];
          },
          FALSE
      ] AUTOSTART;

      ENUM [NON, FULL] SCHEDULE;
      EVENT_TYPE EVENT[];
      RESOURCE_TYPE RESOURCE[];

      UINT32 [1..256] PRIORITY;     // Value range defined
      UINT32 [1..24] ACTIVATION;    // Value range defined

      ENUM [
            SENT
            {
                  MESSAGE_TYPE MESSAGE;
                  BOOLEAN WITHOUTCOPY;
                  STRING ACCESSNAME;
            },
            RECEIVED
            {
                  MESSAGE_TYPE MESSAGE;
                  BOOLEAN WITHOUTCOPY;
                  STRING ACCESSNAME;
            }
      ] ACCESSOR[];

      /* Implementation specific attributes added */
      UINT32 STACKSIZE = 16;
      ENUM [STACKINTERNAL, STACKEXTERNAL] STACKTYPE = STACKINTERNAL;
      MESSAGE_TYPE MESSAGESENT[];
      MESSAGE_TYPE MESSAGERECEIVED[];
      ENUM WITH_AUTO [MINIMUM, MAXIMUM] OPTIMIZE = AUTO;
};
```

```
ISR
{
   UINT32 [1, 2] CATEGORY;
   RESOURCE_TYPE RESOURCE[];
   ENUM [
         SENT
         {
               MESSAGE_TYPE MESSAGE;
               STRING ACCESSNAME;
         },
         RECEIVED
         {
               MESSAGE_TYPE MESSAGE;
               STRING ACCESSNAME;
         }
   ] ACCESSOR[];

   /* Implementation specific attributes added */
   UINT32 STACKSIZE = 32;
   UINT32 [0,1,3,4,7] ISRPRIORITY = 1: "Only ISR-priorities 0,1,3,4,7
   are supported by MySpecificImplementation";
   BOOLEAN [
      TRUE
      {
         TASK_TYPE TASK[];
      }: "and true means true",
      FALSE
      {
      }: "false means false"
   ] taskNotification= FALSE;

};

COUNTER
{
   UINT32 MINCYCLE;
   UINT32 MAXALLOWEDVALUE;
   UINT32 TICKPERBASE;
};

ALARM
{
   COUNTER_TYPE COUNTER;
   ENUM [
      ACTIVATETASK
      {
         TASK_TYPE TASK;
      }: "Task to be activated",

      SETEVENT
      {
         TASK_TYPE TASK;
         EVENT_TYPE EVENT;
      }: "Event to be set"

      ALARMCALLBACK
      {
         STRING ALARMCALLBACKNAME;
      }: "Callback to be called"
   ] ACTION;

   BOOLEAN [
      TRUE {
         UINT32 ALARMTIME;
         UINT32 CYCLETIME;
```

```
            APPMODE_TYPE APPMODE[];
              },
         FALSE
      ] AUTOSTART;

      /* Implementation specific attribute added */
      BOOLEAN START = FALSE: "Automatic start of alarm on system start";
      BOOLEAN CYCLIC = FALSE: "Cyclic alarm";
   };

   EVENT
   {
      UINT32 WITH_AUTO [1..0xff] MASK; // Value range defined
   };

   RESOURCE {
      ENUM [
         STANDARD,
         LINKED {
             RESOURCE_TYPE LINKEDRESOURCE;
         },
         INTERNAL
      ] RESOURCEPROPERTY;

};

   MESSAGE
   {
      ENUM [
            UNQUEUED,
            QUEUED {UINT64 QUEUEDEPTH;}
      ] TYPE;
      STRING CDATATYPE;
      ENUM [
            NONE,
            ACTIVATETASK
            {
                  TASK_TYPE TASK;
            },
            SETEVENT
            {
                  TASK_TYPE TASK;
                  EVENT_TYPE EVENT;
            },
            CALLBACK
            {
                  STRING CALLBACKNAME;
            },
            FLAG
            {
                  STRING FLAGNAME;
            }
      ] ACTION[]; // action to perform if message is sent or received
   };

   COM
   {
      BOOLEAN USEMESSAGERESOURCE;
      BOOLEAN USEMESSAGESTATUS;
   };

   NM
   {
   };
};
```

```
// "Sample application definition";
CPU Sample_CPU1
{
   OS StdOS
   {
      STATUS = STANDARD;
      STARTUPHOOK = TRUE;
      ERRORHOOK = TRUE;
      SHUTDOWNHOOK = TRUE;
      PRETASKHOOK = FALSE;
      POSTTASKHOOK = FALSE;
      USEGETSERVICEID = FALSE;
      USEPARAMETERACCESS = FALSE;
   };
   OS StdOS
   {
      NON_SUSPENDED_TASKS = 4;
      ClockFrequency = 10.0: "Frequency in MHz";  // float and description
   };
   COM StdCOM
   {
      USEMESSAGERESOURCE= FALSE;
      USEMESSAGESTATUS= TRUE;
   };
   ISR myTimerInterrupt
   {
      // ISR STACKSIZE default value
      CATEGORY = 2;
      ISRPRIORITY = 3;
   };
   ISR myExtInterrupt
   {
      CATEGORY = 2;
      ISRPRIORITY = 4;
      RESOURCE = MsgAccess;
      taskNotification= TRUE
      {
         TASK= task1;
         TASK= task2;
      };
      ACCESSOR = RECEIVED {
            MESSAGE= anyMessage2;
            ACCESSNAME= "anyMessage2Buffer";
      };
   };

   TASK TaskSND
   {
      AUTOSTART = FALSE;
      PRIORITY = 3;
      ACTIVATION = 1;
      SCHEDULE = FULL;
      RESOURCE = ResMsgAccess;
      ACCESSOR = SENT {
            MESSAGE= newInfos;
            WITHOUTCOPY= TRUE;
            ACCESSNAME= "newInfosBuffer";
      };
   };
   TASK TaskRCV
   {
      AUTOSTART = FALSE;
      PRIORITY = 1;
      ACTIVATION = 1;
      SCHEDULE = FULL;
```

```
        RESOURCE =  MsgAccess;
    };
```

```
TASK TaskProd
{
   AUTOSTART = FALSE;
   PRIORITY = 2;
   ACTIVATION = 1;
   SCHEDULE = FULL;
   EVENT = timeEvent;
};
TASK TaskCons
{
   AUTOSTART = FALSE;
   PRIORITY = 4;
   ACTIVATION = 1;
   SCHEDULE = NON;
};
EVENT timeEvent
{
   MASK = AUTO;
};
RESOURCE MsgAccess
{
   RESOURCEPROPERTY = STANDARD;
};

COUNTER SystemTimer
{
   MAXALLOWEDVALUE = 65535;
   TICKSPERBASE = 10;
   MINCYCLE = 1;
};

COUNTER MsgCounter
{
   MAXALLOWEDVALUE = 6;
   TICKSPERBASE = 1;
   MINCYCLE = 10;
};

ALARM MsgAlarm
{
   COUNTER = SystemTimer;
   ACTION = ACTIVATETASK
   {
      TASK = TaskSND;
   };
   AUTOSTART = FALSE;
};

ALARM ProdAlarm
{
   COUNTER = SystemTimer;
   ACTION = SETEVENT
   {
      TASK = TaskProd;
      EVENT = timeEvent;
   };
   AUTOSTART = FALSE;
};

ALARM EvMsgAlarm
{
   COUNTER = MsgCounter;
   ACTION = ACTIVATETASK
   {
      TASK = TaskCons;
};
        AUTOSTART = FALSE;
    };
```

```
   TASK TaskProd
   {
      STACKSIZE = 24;
   };

   TASK TaskCons
   {
      STACKTYPE = STACKEXTERNAL;
      STACKSIZE = 8;
   };

   MESSAGE newInfos   {
      TYPE = UNQUEUED;
      CDATATYPE = "infoStruct";
      ACTION= SETEVENT {
            TASK = TaskProd;
            EVENT = timeEvent;
      };
      ACTION= CALLBACK {
          CALLBACKNAME= "msgReceived1";
      };
   };
}: "This CPU is intended to run the Sample application";
```

5.5 Generator hints

All topics concerning generator hints are not part of the specification. They are recommendations.

5.5.1 Generator interface

Recommendation for parameters of system generator

- parameter -a for accept unknown attributes (i.e. ignore attributes which are defined in the implementation-specific part of OIL but for which the generator has no rule)
- parameter -i for include paths
- parameter -f for command file
- parameter -r for generating resource statistics
- parameter -v for version
- parameter -t for test/verify

From the user point of view, all implementation-specific switches (of the generator) should be attributes of the matching OIL objects. This would allow the user to place all the implementation-specific information in the OIL file and not into command-line parameters.

5.5.2 Resource usage statistics

The generator should provide all resources of the operating system used by the application (e.g. number of tasks, priorities, ...) to the user.

5.5.3 Naming convention for OIL files

For ease of use the main OIL file should have the file extension .OIL. The extensions for other files that are included in the main OIL file are free.

6 Changes in specifications

6.1 Changes from specification 1.0/2.0 to 2.1

The specifications 1.0/2.0 were no official versions, so no change description is provided.

6.2 Changes from specification 2.1 to 2.2

6.2.1 Resources

According to the OS specification 2.1, resources may be used in interrupt service routines. A standard attribute to reference a resource object was added.

6.2.2 Messages

The OS specification 2.1 referes to OSEK COM as two additional conformance classes for local message handling. Standard attributes for messages were added. References from TASKs and ISRs to messages were added, too.

6.2.3 COM

The COM object got two standard attributes. Additionaly it was defined, that the COM object may be defined only once.

6.3 Changes from specification 2.2 to 2.3

The following changes were made to support the new features of the OS specification 2.2.

6.3.1 ALARM

An AUTOSTART attribute was added to the ALARM object.

The ACTION attribute was amended by a third value ALARMCALLBACK.

6.3.2 ISR

The ISR category 3 was removed.

6.3.3 RESOURCE

The RESOURCEPROPERTY attribute was introduced to handle the new concepts of linked and internal resources.

6.3.4 TASK

The AUTOSTART attribute was modified to support different application modes.

6.3.5 OS

New attributes USEGETSERVICEID and USEPARAMETERACCESS.

7 Index

8 History

Version	Date	Remarks	
2.0	December 16, 1997	Authors:	
Published as		Jürgen Aminger	IBM GmbH
Recommendation		Vladimir Belov	Motorola SPRL
		Jürgen Betzelt	Daimler-Benz AG
		Volker Ebner	Vector Informatik
		Bob France	Motorola SPS
		Gerhard Göser	Siemens Automotive SA
		Martin Huber	Daimler-Benz AG
		Adam Jankowiak	Daimler-Benz AG
		Winfried Janz	Vector Informatik
		Helmar Kuder	Daimler-Benz AG
		Ansgar Maisch	University of Karlsruhe
		Rainer Müller	IBM GmbH
		Salvatore Parisi	Centro Ricerche Fiat
		Jochem Spohr	ATM Computer GmbH
		Stephan Steinhauer	Daimler-Benz AG
		Karl Westerholz	Siemens Semiconductors
		Andree Zahir	ETAS GmbH & Co. KG

Version	Date	Remarks	
2.1	June 30, 1999	Authors:	
Specification		Michael Barbehenn	Motorola
		Irina Bratanova	Motorola
		Manfred Geischeder	BMW
		Gerhard Göser	Siemens Automotive
		Andrea Hauth	3Soft
		Adam Jankowiak	DaimlerChrysler
		Winfried Janz	Vector Informatik
		Helmar Kuder	DaimlerChrysler
		Stefan Schimpf	ETAS
		Markus Schwab	Infineon
		Carsten Thierer	University of Karlsruhe
		Hans-Christian Wense	Motorola
		Andree Zahir	ETAS

Version	Date	Remarks	
2.2	July 4, 2000	Authors:	
Specification		Manfred Geischedder	BMW
		Irina Bratanova	Motorola
		Winfried Janz	Vector
		Reiner Kriesten	IIIT, Uni Karlsruhe
		Jochem Spohr	IMH
		Peter Großhans	IMH
		Walter Koch	Siemens
		Hartmut Hörner	Vector
2.3	August 28, 2001	Authors:	
Specification		OS working group	ISO