

# **Cybernetics Oriented Language (CYBOL)**

An interpretable Knowledge Modelling- and Programming Language



Christian Heller

# Cybernetics Oriented Language (CYBOL)

An interpretable Knowledge Modelling- and Programming Language

Version 2.0, Draft 2007-07-31



---

Ilmenau

## Cataloging-in-Publication Data

Christian Heller.

Cybernetics Oriented Language (CYBOL):

An interpretable Knowledge Modelling- and Programming Language

Version 2.0, Draft 2007-07-31

Ilmenau: Tux Tax, 2007

Information about this specification

<http://www.cybop.net>, <http://www.tuxtax.de>

Copyright © 2002-2007. Christian Heller. All rights reserved.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts and with no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

### Trademark Credits

Most of the software-, hardware- and product names used in this document are also trademarks or registered trademarks of their respective owners.

### Donations

Companies using CYBOP ideas, CYBOL or CYBOI on a grand scale are asked to notify the author <[christian.heller@tuxtax.de](mailto:christian.heller@tuxtax.de)> and to consider donating some of their sales revenues, which will be used exclusively for the CYBOP and Res Medicinae free software projects.

Made in Germany, Europe, Planet Earth

To all Coders who stick to The Hacker Manifesto  
and devote their Playful Cleverness with great Enthusiasm  
to the Free Software Code Base of the Open Source Community



# Contents

<b>Preface</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Theory . . . . .	1
1.2 Analogy . . . . .	2
<b>2 Definition</b>	<b>3</b>
2.1 Syntax . . . . .	3
2.2 Vocabulary . . . . .	4
2.2.1 Document Type Definition . . . . .	4
2.2.2 XML Schema Definition . . . . .	6
2.2.3 Extended Backus Naur Form . . . . .	6
2.3 Semantics . . . . .	9
2.3.1 Attributes . . . . .	9
2.3.2 Tags . . . . .	10
2.3.3 Tag-Attribute Swapping . . . . .	11
<b>3 State Models</b>	<b>13</b>
3.1 User Interface . . . . .	13
3.1.1 Textual User Interface . . . . .	13
3.1.2 Graphical User Interface . . . . .	19
3.1.3 Web User Interface . . . . .	24
<b>4 Logic Models</b>	<b>29</b>
4.1 Bit Manipulation . . . . .	29
4.1.1 Shift . . . . .	29

---

4.1.2	Rotate	30
4.1.3	Set Bit	32
4.1.4	Reset Bit	33
4.1.5	Get Bit	34
4.2	Boolean Logic	35
4.2.1	NOT	35
4.2.2	NEG	36
4.2.3	AND	37
4.2.4	OR	38
4.2.5	XOR	39
4.3	Program Flow	40
4.3.1	Branch	40
4.3.2	Loop	42
4.3.3	Count	42
4.3.4	Get	44
4.4	Comparison	45
4.4.1	Compare	45
4.5	Arithmetic	47
4.5.1	Add	47
4.5.2	Subtract	48
4.5.3	Multiply	50
4.5.4	Divide	51
4.6	Memory Management	52
4.6.1	Create	52
4.6.2	Destroy	54
4.6.3	Copy	54
4.6.4	Move	55
4.7	Lifecycle Management	56
4.7.1	Startup	56
4.7.2	Shutdown	58
4.7.3	Exit	58
4.8	Communication	59
4.8.1	Send	59
4.8.2	Receive	62
4.8.3	Interrupt	65



---

4.9	Shell Commands . . . . .	65
4.9.1	Archive File . . . . .	65
4.9.2	Copy File . . . . .	67
4.9.3	Execute File . . . . .	68
4.9.4	List File . . . . .	68
<b>5</b>	<b>Examples</b>	<b>71</b>
5.1	State Examples . . . . .	71
5.1.1	Model-Part Relation . . . . .	71
5.1.2	Meta Properties . . . . .	72
5.1.3	External Resources . . . . .	73
5.1.4	Serialised Model . . . . .	73
5.1.5	Meta Constraints . . . . .	74
5.2	Logic Examples . . . . .	75
5.2.1	Operation Call . . . . .	75
5.2.2	Algorithm Division . . . . .	76
5.2.3	Simple Assignment . . . . .	76
5.2.4	Loop as Operation . . . . .	77
5.2.5	Conditional Execution . . . . .	78
5.3	Special Examples . . . . .	80
5.3.1	Synchronous Execution . . . . .	80
5.3.2	Presentation and Content . . . . .	82
5.3.3	Hello World . . . . .	84
5.4	Inheritance as Property . . . . .	85
5.5	Container Mapping . . . . .	86
5.6	Hidden Patterns . . . . .	87
<b>6</b>	<b>Diagrams</b>	<b>89</b>
<b>7</b>	<b>Appendices</b>	<b>93</b>
7.1	Abbreviations . . . . .	93
7.2	References . . . . .	95
7.3	Figures . . . . .	97
7.4	Tables . . . . .	99
7.5	History . . . . .	101
7.6	Licences . . . . .	103

7.6.1	GNU General Public License . . . . .	103
7.6.2	GNU Free Documentation License . . . . .	111
7.7	Index . . . . .	121

## Preface

## Prologue

After having had completed and published my book on the theory of *Cybernetics Oriented Programming* (CYBOP) [7], the next logical step was to closer inspect and define the features of the *Cybernetics Oriented Language* (CYBOL), in other words: to write a specification. This book tries to achieve this by:

1. explaining the CYBOL syntax, vocabulary and semantics
2. listing all currently interpretable CYBOL keywords
3. giving small CYBOL code examples

CYBOL is a growing language undergoing steady development. Hence, this book will not be the last version of the CYBOL specification. Also, it is not and doesn't claim to be free of mistakes. So, if you find errors of whatever kind or have any helpful ideas or constructive critics [12], then please contribute them to <christian.heller@tuxtax.de> or to the CYBOP developers mailing list <cybop-developers@lists.berlios.de>!

I am currently thinking about writing a third book dealing with the internal architecture of the *Cybernetics Oriented Interpreter* (CYBOI). However, this is an issue for the future.

## Stylistic Means and Notation

The language of choice in this document is *British English*, more precisely known as *Commonwealth English*. Exceptions are citations or proper names like *Unified Modeling Lan-*

*guage*, stemming from *American English* sources. (In Oxford English, *Modelling* would be written with double letter *l*).

Correctly, masculine *and* feminine forms are used in a work. When describing a person's address, for example, one would write: *his or her address*. In order to improve readability, and exclusively because of this reason, only masculine forms are used in this work.

Knowledge templates/ models or pieces thereof, as well as CYBOL keywords are displayed in **Typewriter Typeface**. Emphasised words are *italicised*.

Footnotes are not used on purpose. In my opinion, they only interrupt the flow-of-reading. Remarks are placed in context instead, sometimes enclosed in parentheses.

The *Appendices* (chapter 7) contain used abbreviations, references to literature and the usual lists of figures and tables, as well as the document's history and licences in full text. A glossary was omitted since this document does not want to be a lexicon. All terms are explained at their first appearance in the text. Caution! The page numbers behind an index entry at the end of this document refer to the *Beginning* of the section in which the entry appeared.

*Kostrzyn Wielkopolska, Poland*

*July 2007*

*Christian Heller <christian.heller@tuxtax.de>*

# 1 Introduction

The *Cybernetics Oriented Language* (CYBOL) is an interpretable knowledge modelling- and programming language. It is very flexible, has a simple syntax and is easy to learn. Hence, not only classical software designers- and developers, but also analysts and domain experts may have an interest in CYBOL.

Its simplicity is based on the fact that just one major concept needs to be understood: that of *Hierarchy*. Well, in fact CYBOL applies it in form of a *Double Hierarchy*, but is this not very difficult to grasp, as the code examples later in this document will show.

## 1.1 Theory

The theory behind CYBOL is called *Cybernetics Oriented Programming* (CYBOP). It describes the general concepts, software architecture and development principles that justify the existence of CYBOL. Besides CYBOL as language, CYBOP suggests the *Cybernetics Oriented Interpreter* (CYBOI).

Considering an overall computer system architecture, *CYBOI* as low-level software system is situated between the application knowledge existing in form of *CYBOL* templates and the *Hardware* controlled by an *Operating System* (OS), as is shown in figure 1.1. *CYBOI* as *active* system process interprets the *passive* application knowledge provided in form of CYBOL files.

Design-time knowledge resides in *CYBOL Knowledge Templates* (files). While being instantiated, it gets transferred into runtime *Knowledge Models* that reside in a computer's *Random Access Memory* (RAM).

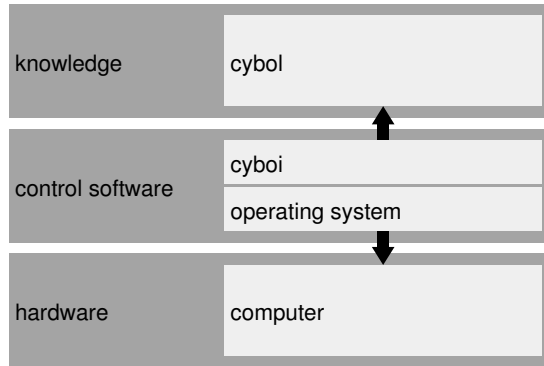


Figure 1.1: CYBOL Interpretation

## 1.2 Analogy

There are analogies to other systems run by language interpretation. Table 1.1 shows that between the *Java*- and *CYBOP* world. Both are based on a programming theory, have a language and interpreter, the latter sometimes being called a *Virtual Machine* (VM).

Criterion	Java World	CYBOP World
Theory	OOP in Java	CYBOP
Language	Java	CYBOL
Interpreter	Java VM	CYBOI

Table 1.1: Analogy between the Java- and CYBOP World

CYBOI provides low-level, platform-dependent system functionality, close to the OS, together with a unified knowledge schema which allows CYBOL applications to be truly portable, well extensible and easier to program, because developers need to concentrate on domain knowledge only. Since CYBOI may interpret CYBOL sources *live* at system runtime, without the need for previous compilation (as in Java), changes to CYBOL sources can get into effect right away, without having to restart the system.

## 2 Definition

This chapter defines the *Syntax*, *Vocabulary* and *Semantics* of the CYBOL language.

As already mentioned in chapter 1, CYBOL is based upon *two* kinds of hierarchies. One of them is representing *Whole-Part* relations (such as a graphical window consisting of a menu bar) and the other the *Meta Data* which a whole keeps about its parts (such as the size or colour of the menu bar). More details and the philosophical background are described in [7]. The syntax and semantics of CYBOL as new language must be rich enough to express abstract knowledge models using this kind of double hierarchy.

### 2.1 Syntax

CYBOL's syntax (grammar with rules for combining terms and symbols) is based on the well-known *Extensible Markup Language* (XML) [17]. It has a clear text representation, is flexible, extensible and easy to use.

To mention just two of the syntactical elements of XML, *Tag* and *Attribute* are considered here shortly. Tags are special, arbitrary keywords that have to be defined by the system working with an XML document. Attributes keep additional information about the contents enclosed by two tags. Two examples:

```
<tag attribute="value">
  contents
</tag>
```

```
<tag attribute1="value" attribute2="content"/>
```

A CYBOL knowledge template (XML document) carries a name and can thus represent a *Discrete Item*. Being a *Compound*, the template consists of parts – and, it can link to other templates (files) treated as its parts. That way, a whole hierarchy can be formed. Tag attributes can keep additional information about the linked parts. Most importantly, the hierarchical structure is based on tags, which may be used to store meta data about a part.

CYBOL, finally, is XML *plus* a defined set of tags, attributes and values used to structure and link documents meaningfully.

## 2.2 Vocabulary

The *Vocabulary* is what fills a language with life. It delivers the *Terms* and *Symbols* that are combined after the rules of a syntax.

XML allows to define and exchange the whole vocabulary of a language. It offers two ways in which a list of legal elements can be defined: The traditional *Document Type Definition* (DTD) and the more modern *XML Schema Definition* (XSD). Besides the vocabulary, DTD and XSD define the structure of an XML document and allow to typify, constrain and validate items. In addition to DTD and XSD, the *Extended Backus Naur Form* (EBNF) of CYBOL is given following.

### 2.2.1 Document Type Definition

A DTD represents the type definition of an XML document. It consists of a set of *Markup Tags* and their *Interpretation* [8]. DTDs can be declared inline, within a document, or as an external reference [15]. Figure 2.1 shows the DTD of the CYBOL language.

Following the pure hierarchical structure of CYBOL, it would actually suffice to use a DTD as simple as the one shown in figure 2.2. Since the three elements *part*, *property* and *constraint* (compare figure 2.1) have the same list of required attributes, they could be summarised under the name *part*, for example. Because the structure of a CYBOL model is non-ambiguous, the meaning of its elements can be guessed from their position within the model.



```
<!ELEMENT model (part*)>
<!ELEMENT part (property*)>
<!ELEMENT property (constraint*)>
<!ELEMENT constraint EMPTY>

<!ATTLIST part
  name CDATA #REQUIRED
  channel CDATA #REQUIRED
  abstraction CDATA #REQUIRED
  model CDATA #REQUIRED>
<!ATTLIST property
  name CDATA #REQUIRED
  channel CDATA #REQUIRED
  abstraction CDATA #REQUIRED
  model CDATA #REQUIRED>
<!ATTLIST constraint
  name CDATA #REQUIRED
  channel CDATA #REQUIRED
  abstraction CDATA #REQUIRED
  model CDATA #REQUIRED>
```

Figure 2.1: Recommended CYBOL DTD

```
<!ELEMENT part (part*)>

<!ATTLIST part
  name CDATA #REQUIRED
  channel CDATA #REQUIRED
  abstraction CDATA #REQUIRED
  model CDATA #REQUIRED>
```

Figure 2.2: Simplified CYBOL DTD

For the purpose of expressing knowledge in accordance with the schema suggested by CYBOP [13], a CYBOL knowledge template (file) does not need to have a root element. The file name clearly identifies it. For reasons of XML conformity, however, an extra root element called *model* was defined (figure 2.1). And for reasons of better readability and programmability, the three kinds of embedded elements were given distinct names.

### 2.2.2 XML Schema Definition

*XML Schema* is an XML-based alternative to DTD [15], and XSD is its definition language. Figure 2.4 shows the XSD of the CYBOL language.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema' targetNamespace='http://www.cybop.net'
  xmlns='http://www.cybop.net' elementFormDefault='qualified'>
  <xs:element name='part'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='part' minOccurs='0' maxOccurs='unbounded' />
      </xs:sequence>
      <xs:attribute name='name' type='xs:string' use='required' />
      <xs:attribute name='channel' type='xs:string' use='required' />
      <xs:attribute name='abstraction' type='xs:string' use='required' />
      <xs:attribute name='model' type='xs:string' use='required' />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 2.3: Simplified CYBOL XSD

Again, a simplified version of that XSD could be created (figure 2.3). But for reasons explained before, the recommended XSD is the one shown in figure 2.4.

### 2.2.3 Extended Backus Naur Form

The EBNF adds regular expression syntax to the *Backus Naur Form* (BNF) notation [1], in order to allow very compact specifications [9]. Figure 2.5 shows the EBNF of the CYBOL language.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.cybop.net"
  xmlns="http://www.cybop.net" elementFormDefault="qualified">
  <xs:element name="model">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="part" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="part">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="property" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="channel" type="xs:string" use="required"/>
      <xs:attribute name="abstraction" type="xs:string" use="required"/>
      <xs:attribute name="model" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="property">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="constraint" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="channel" type="xs:string" use="required"/>
      <xs:attribute name="abstraction" type="xs:string" use="required"/>
      <xs:attribute name="model" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="constraint">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="channel" type="xs:string" use="required"/>
      <xs:attribute name="abstraction" type="xs:string" use="required"/>
      <xs:attribute name="model" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 2.4: Recommended CYBOL XSD

```

CYBOL      = '<model>'
              {part}
              '</model>';

part       = '<part ' attributes '\>' |
              '<part ' attributes '>'
              {property}
              '</part>';

property   = '<property ' attributes '\>' |
              '<property ' attributes '>'
              {constraint}
              '</property>';

constraint = '<constraint ' attributes '\>';

attributes = name_attribute channel_attribute abstraction_attribute model_attribute

name_attribute      = 'name=' name '';
channel_attribute   = 'channel=' channel '';
abstraction_attribute = 'abstraction=' abstraction '';
model_attribute     = 'model=' model '';

name      = description_sign;
channel   = description_sign;
abstraction = description_sign;
model     = value_sign;

description_sign = { ( letter | number ) };
value_sign      = { ( letter | number | other_sign ) };

letter      = small_letter | big_letter;
small_letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' |
              'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' |
              'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' |
              'v' | 'w' | 'x' | 'y' | 'z';
big_letter  = 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' |
              'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' |
              'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' |
              'V' | 'W' | 'X' | 'Y' | 'Z';
other_sign  = ',' | '.' | '/' | '+' | '-' | '*';
number     = '0' | '1' | '2' | '3' | '4' |
              '5' | '6' | '7' | '8' | '9';

```

Figure 2.5: CYBOL in EBNF

## 2.3 Semantics

The meaning expressed by terms and sentences is their *Semantics* [2].

CYBOL files can be used to model *State Knowledge* (like a graphical window or a person's address) and *Logic Knowledge* (like an operation or algorithm or workflow) [7]. In both cases, the *same* syntax (document structure) with *identical* vocabulary (XML tags and -attributes) is applied. It is the attribute *Values* that make a difference in meaning.

The double hierarchy mentioned before is realised in CYBOL knowledge templates by using XML *Attributes* for representing the whole-part hierarchy, and XML *Tags* for representing the additional meta data that a whole model keeps about its part models.

### 2.3.1 Attributes

Normally, an XML *Attribute* keeps meta information about the contents of an XML *Tag*. In CYBOL, however, three attributes keep meta information about a fourth attribute. The attributes, altogether, are:

- name
- channel
- abstraction
- model

The attribute of greatest interest is *model*. It contains a model either directly, or a path to one. The *channel* attribute indicates whether the *model* attribute's value is to be read from:

- inline
- file

The *abstraction* attribute specifies how to interpret the model pointed to by the *model* attribute's value. A model may be given in formats like for example:

- compound (a state- or logic compound model encoded in CYBOL format)
- operation (a primitive logic model)
- character

- double
- integer
- boolean

The *name* attribute, finally, provides the referenced model with an identifier that has to be unique within the *Whole* model the *Part* model belongs to.

While the interpretation of the *model* attribute's value depends on the *channel*- and *abstraction* attributes, the other three attributes (*name*, *channel*, *abstraction*) themselves always get interpreted as string of characters.

### 2.3.2 Tags

There are many kinds of meta data besides the above-mentioned attributes, that may be known about a model. These are given as special XML tags called *property* and *constraint*. As defined in section 2.2, a CYBOL knowledge template may use four kinds of XML tags:

- model
- part
- property
- constraint

The *model* tag appears just once. It is the root node which makes a CYBOL knowledge template a valid XML document.

Of actual interest are the *part* tags. They identify the models that the *whole* model described by the CYBOL knowledge template consists of.

A *whole* model may know a lot more about its *part* models, than is given by a part model's XML attributes. A spatial state model may know about the *position* and *size* of its parts, in space. A temporal model (such as a workflow) may have to know about the *position* of its parts in time, in order to be able to execute them in the correct order. Further, the temporal model needs to know about the *input/output* (i/o) state models which are to be manipulated by the corresponding logic operation (part model). The number of parts within a whole (compound) model may be limited. And so on. These additional information are provided by *property* tags whose number is conceptually unlimited.

Not only parts need additional meta data; properties may need such data, too. The position or size as properties of a part may have to be constrained to certain values, such as a *minimum* or *maximum*. The values of the *colour* property of a part model may have to be chosen out of a pre-defined set called *choice*. Data of that kind are stated in *constraint* tags.

### 2.3.3 Tag-Attribute Swapping

CYBOL swaps the meaning attributes and tags traditionally have in XML documents, where tags represent elements that may be nested infinitely and attributes hold additional (meta) data about a tag. Following an example of how CYBOL might have looked that way:

```
<model>
  <part>
    <name="title"/>
    <channel="inline"/>
    <abstraction="character"/>
    <model="Res Medicinae"/>
  </part>
  <part layout="compass" position="north">
    <name="menu_bar"/>
    <channel="file"/>
    <abstraction="cybol"/>
    <model="gui/menu_bar.cybol"/>
  </part>
</model>
```

The current final specification of CYBOL, on the contrary, uses attributes to define a nested element (part) and tags to give properties (meta information) about such a nested element, in the following way:

```
<model>
  <part name="title" channel="inline" abstraction="character" model="Res Medicinae"/>
  <part name="menu_bar" channel="file" abstraction="cybol" model="gui/menu_bar.cybol">
    <property name="layout" channel="inline" abstraction="character" model="compass"/>
    <property name="position" channel="inline" abstraction="character" model="north"/>
  </part>
</model>
```

This is because:

1. the number of attributes specifying a part in CYBOL is fixed, whereas the number of tags specifying a property of a part is not, and the number of XML tags is easier extensible than that of attributes;
2. that way it is also possible to specify a part without any properties in just one CYBOL code line, while otherwise four tags would always have to be given;
3. not only a part may be nested (consist of smaller parts), but also a property may be (for example a position consisting of three coordinates given as parts), which necessitates the four standard attributes to be given for properties and constraints as well.



## 3 State Models

### 3.1 User Interface

A *User Interface* (UI) provides functionality by which a user can communicate with a software system.

#### 3.1.1 Textual User Interface

*Textual User Interface* (TUI) is a synonym for character-based UI. Historically, the TUI (besides the simple command line) was the first kind of UI for computers. It mostly offers a menu with a list of possible choices that can be activated via the pressing of a special button on the keyboard. Figure 3.1 illustrates a typical, menu-based TUI.

#### Example

```
<part name="exit_menu_item" channel="inline" abstraction="character" model="Exit">
  <property name="position" channel="inline" abstraction="integer" model="5,10,0"/>
  <property name="size" channel="inline" abstraction="integer" model="60,1,1"/>
  <property name="background" channel="inline" abstraction="character" model="blue"/>
  <property name="foreground" channel="inline" abstraction="character" model="white"/>
  <property name="bold" channel="inline" abstraction="boolean" model="true"/>
  <property name="enter" channel="inline" abstraction="knowledge" model=".app.logic.enter"/>
  <property name="previous" channel="inline" abstraction="knowledge" model=".app.tui.about_menu_item"/>
  <property name="next" channel="inline" abstraction="knowledge" model=".app.tui.start_menu_item"/>
  <property name="arrow_up" channel="inline" abstraction="knowledge" model=".app.logic.arrow_up"/>
  <property name="arrow_down" channel="inline" abstraction="knowledge" model=".app.logic.arrow_down"/>
</part>
```

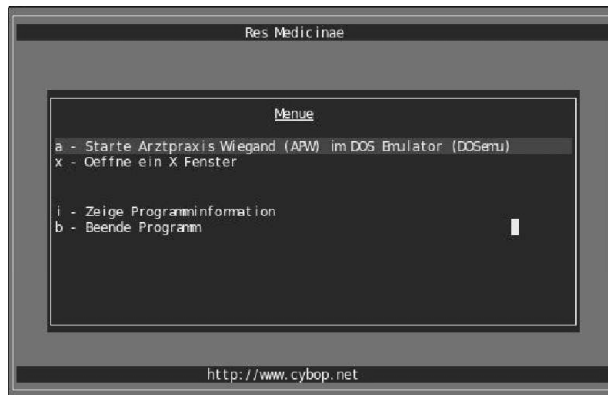


Figure 3.1: Textual User Interface

### Position Property

This property specifies the TUI element's origin.

*required*

```
name='position'  
abstraction='integer'  
model=x, y, z coordinates
```

### Size Property

This property specifies the TUI element's extension.

*required*

```
name='size'  
abstraction='integer'  
model=x, y, z extensions
```

## Background Property

This property specifies the background colour of the TUI.

*required*

name='background'

abstraction='character'

model='black' | 'red' | 'green' | 'yellow' | 'blue' | 'magenta' | 'cobalt' | 'white'

## Foreground Property

This property specifies the foreground colour of the TUI.

*required*

name='foreground'

abstraction='character'

model='black' | 'red' | 'green' | 'yellow' | 'blue' | 'magenta' | 'cobalt' | 'white'

## Border Property

This property specifies the kind of border of the TUI.

*optional*

name='border'

abstraction='character'

model='line' | 'round\_line' | 'double\_line'

## Hidden Property

This property specifies whether or not to hide the TUI element.

*optional*

name='hidden'

abstraction='boolean'

model='true' | 'false'

### **Inverse Property**

This property specifies whether or not to display the TUI element in inverse colours.

*optional*

```
name='inverse'  
abstraction='boolean'  
model='true' | 'false'
```

### **Blink Property**

This property specifies whether or not the TUI element should blink.

*optional*

```
name='blink'  
abstraction='boolean'  
model='true' | 'false'
```

### **Underline Property**

This property specifies whether or not to underline the TUI element's text.

*optional*

```
name='underline'  
abstraction='boolean'  
model='true' | 'false'
```

### **Bold Property**

This property specifies whether or not to display the TUI element's text in bold font.

*optional*

```
name='bold'  
abstraction='boolean'  
model='true' | 'false'
```

### Focus Property

This property points to the TUI element (knowledge model) having focus. It is important to find out which TUI element a key press event relates to. The focus of a number of part elements should always be held by their corresponding whole (compound) element.

*optional*, only if TUI element should be able to react to button press events

```
name='focus'  
abstraction='knowledge' | 'encapsulated'  
model=logic knowledge model
```

### Previous Property

This property points to the previous TUI element (knowledge model) owning focus.

*optional*, only if TUI element should have a predecessor that may own the focus

```
name='previous'  
abstraction='knowledge' | 'encapsulated'  
model=logic knowledge model
```

### Next Property

This property points to the next TUI element (knowledge model) receiving focus.

*optional*, only if TUI element should have a successor that may own the focus

```
name='next'  
abstraction='knowledge' | 'encapsulated'  
model=logic knowledge model
```

### Enter Property

This property specifies the logic knowledge model to be executed if the *Enter* button is pressed while the TUI element has focus.

*optional*, only if TUI element should react to button press event; a prerequisite is that the TUI element's *focus* property value is *true*

```
name='enter'  
abstraction='knowledge' | 'encapsulated'  
model=logic knowledge model
```

### Escape Property

This property specifies the logic knowledge model to be executed if the *Esc* button is pressed while the TUI element has focus.

*optional*, only if TUI element should react to button press event; a prerequisite is that the TUI element's *focus* property value is *true*

```
name='escape'  
abstraction='knowledge' | 'encapsulated'  
model=logic knowledge model
```

### Arrow Up Property

This property specifies the logic knowledge model to be executed if the *arrow-up* button is pressed while the TUI element has focus.

*optional*, only if TUI element should react to button press event; a prerequisite is that the TUI element's *focus* property value is *true*

```
name='arrow.up'  
abstraction='knowledge' | 'encapsulated'  
model=logic knowledge model
```

### Arrow Down Property

This property specifies the logic knowledge model to be executed if the *arrow-down* button is pressed while the TUI element has focus.

*optional*, only if TUI element should react to button press event; a prerequisite is that the TUI element's *focus* property value is *true*

```
name='arrow.down'  
abstraction='knowledge' | 'encapsulated'
```

```
model=logica knowledge model
```

### Arrow Left Property

This property specifies the logic knowledge model to be executed if the *arrow-left* button is pressed while the TUI element has focus.

*optional*, only if TUI element should react to button press event; a prerequisite is that the TUI element's *focus* property value is *true*

```
name='arrow_left'  
abstraction='knowledge' | 'encapsulated'  
model=logica knowledge model
```

### Arrow Right Property

This property specifies the logic knowledge model to be executed if the *arrow-right* button is pressed while the TUI element has focus.

*optional*, only if TUI element should react to button press event; a prerequisite is that the TUI element's *focus* property value is *true*

```
name='arrow_right'  
abstraction='knowledge' | 'encapsulated'  
model=logica knowledge model
```

## 3.1.2 Graphical User Interface

A *Graphical User Interface* (GUI) is mostly based on so-called graphical *Windows* which may overlap, or be ordered side-by-side on a screen. Figure 3.2 illustrates a typical GUI.

### Example

```
<part name="menu_bar" channel="file" abstraction="compound" model="gui/menu_bar.cybol">  
  <property name="shape" channel="inline" abstraction="character" model="rectangle"/>  
  <property name="layout" channel="inline" abstraction="character" model="compass"/>  
  <property name="cell" channel="inline" abstraction="character" model="north"/>  
  <property name="size" channel="inline" abstraction="integer" model="600,40,1"/>
```

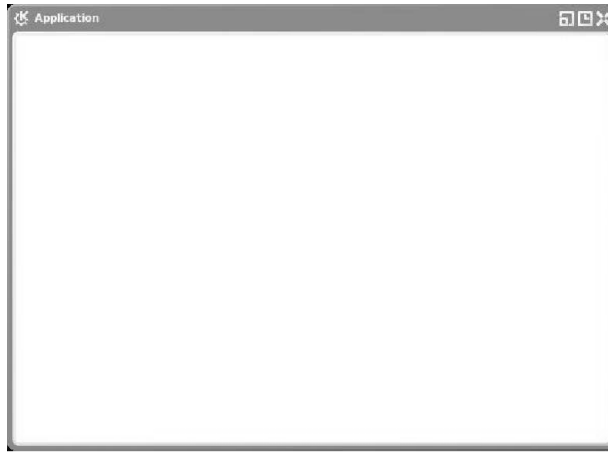


Figure 3.2: Graphical User Interface

```
<property name="foreground" channel="inline" abstraction="rgb" model="0,0,0"/>
<property name="right_press" channel="inline" abstraction="knowledge" model=".app.logic.action"/>
</part>
```

### Shape Property

This property specifies the geometrical shape of the GUI.

*required*

```
name='shape'
abstraction='character'
model='rectangle' | 'circle' | 'polygon'
```

### Layout Property

This property specifies the kind of layout of the GUI.

*required*



```
name='layout'  
abstraction='character'  
model='root' | 'coordinates' | 'compass'
```

### Cell Property

This property specifies the cell ordering, if *compass* layout is used.

*optional*, only if *layout* property is *compass*

```
name='cell'  
abstraction='character'  
model='north' | 'south' | 'west' | 'east' | 'centre'
```

### Position Property

This property specifies the GUI element's origin.

*optional*, only if *layout* property is *root* or *coordinates*

```
name='position'  
abstraction='integer'  
model=x, y, z coordinates
```

### Size Property

This property specifies the GUI element's extension.

*required*

```
name='size'  
abstraction='integer'  
model=x, y, z extensions
```

### Background Property

This property specifies the background colour of the GUI.

*optional*

```

name='background'
abstraction='character'
model='black' | 'red' | 'green' | 'yellow' | 'blue' | 'magenta' | 'cobalt' | 'white'

```

### Foreground Property

This property specifies the foreground colour of the GUI.

*optional*

```

name='foreground'
abstraction='character'
model='black' | 'red' | 'green' | 'yellow' | 'blue' | 'magenta' | 'cobalt' | 'white'

```

### Title Property

This property specifies the GUI element's (window's) title.

*optional*, only if *layout* property is *root*

```

name='title'
abstraction='character'
model=window title

```

### Icon Property

This property specifies the GUI element's (window's) icon.

*optional*, only if *layout* property is *root*

```

name='icon'
abstraction='bmp' | 'jpeg' | 'png' | 'gif' | etc.
model=graphic file

```

### Expose Property

This property specifies the logic knowledge model to be executed if the GUI element is exposed, for example shown again after having been hidden before.

*optional*, only if GUI element should react to expose event

```
name='expose'  
abstraction='knowledge' | 'encapsulated'  
model=logica knowledge model
```

### Mouse Over Property

This property specifies the logic knowledge model to be executed if the mouse is moved over the GUI element.

*optional*, only if GUI element should react to mouse event

```
name='mouse_over'  
abstraction='knowledge' | 'encapsulated'  
model=logica knowledge model
```

### Mouse Wheel Property

This property specifies the logic knowledge model to be executed if the mouse wheel is scrolled while the mouse pointer is over the GUI element.

*optional*, only if GUI element should react to mouse event

```
name='mouse_wheel'  
abstraction='knowledge' | 'encapsulated'  
model=logica knowledge model
```

### Left Press Property

This property specifies the logic knowledge model to be executed if the left mouse button is pressed.

*optional*, only if GUI element should react to mouse event

```
name='left_press'  
abstraction='knowledge' | 'encapsulated'  
model=logica knowledge model
```

### Left Release Property

This property specifies the logic knowledge model to be executed if the left mouse button is released.

*optional*, only if GUI element should react to mouse event

```
name='left_release'
abstraction='knowledge' | 'encapsulated'
model=logik knowledge model
```

### Left Click Property

This property specifies the logic knowledge model to be executed if the left mouse button is clicked. A mouse click is the combination of a mouse press- and release event.

*optional*, only if GUI element should react to mouse event

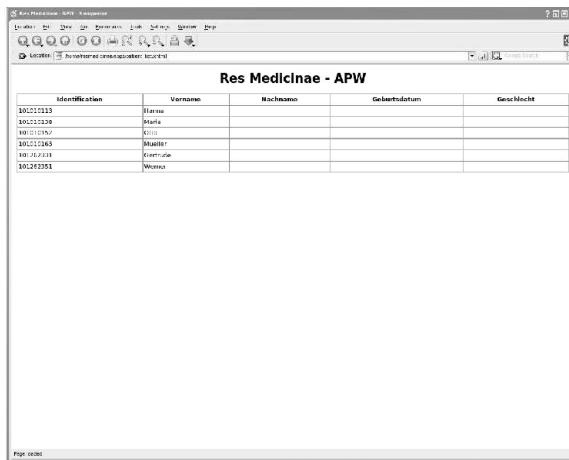
```
name='left_click'
abstraction='knowledge' | 'encapsulated'
model=logik knowledge model
```

## 3.1.3 Web User Interface

A *Web User Interface* (WUI) is what is commonly also known as *Hypertext Markup Language* (HTML) file. HTML files get interpreted by a *Web Browser* which renders the graphical and textual information to display these as HTML page. Figure 3.3 shows an example WUI.

### Example

```
<part name="address_table" channel="file" abstraction="compound" model="wui/table.cybol">
  <property name="tag" channel="inline" abstraction="character" model="table"/>
  <property name="width" channel="inline" abstraction="character" model="100%"/>
  <property name="cellspacing" channel="inline" abstraction="integer" model="5"/>
  <property name="cellpadding" channel="inline" abstraction="integer" model="2"/>
  <property name="border" channel="inline" abstraction="boolean" model="false"/>
</part>
```



Identifikation	Vorname	Nachname	Geburtsdatum	Geschlecht
100111219	Name			
100111209	Marie			
100111187	Karl			
100111189	Maier			
100111118	Karlsson			
100111251	Werner			

Figure 3.3: Web User Interface

### Tag Property

This property specifies the HTML tag to associate with the knowledge model.

*required*

name='tag'

abstraction='character'

model='html' | 'head' | 'meta' | 'body' | 'table' | 'tr' | etc.

### Xmlns Property

This property specifies the XML namespace for the HTML page to be generated from the WUI.

*optional*, only for *html* tag

name='xmlns'

abstraction='character'

model='http://www.w3.org/1999/xhtml'

### **HTTP-Equiv Property**

This property specifies the content type for the HTML page to be generated from the WUI.

*optional*, only for *meta* tag

```
name='http-equiv'  
abstraction='character'  
model='content-type'
```

### **Name Property**

This property specifies the name of a meta data entry for the HTML page to be generated from the WUI.

*optional*, only for *meta* tag

```
name='name'  
abstraction='character'  
model='author'
```

### **Content Property**

This property specifies the content of a meta data entry for the HTML page to be generated from the WUI.

*optional*, only for *meta* tag

```
name='content'  
abstraction='character'  
model='Generated by CYBOI'
```

### **Align Property**

This property specifies the alignment of the HTML heading.

*optional*, only for *h1..6* tag

```
name='align'  
abstraction='character'  
model='left' | 'right' | 'center'
```

### Width Property

This property specifies the width of the HTML table.

*optional*, only for *table* tag

```
name='width'  
abstraction='character'  
model=table width
```

### Cellspacing Property

This property specifies the spacing between the cells of the HTML table.

*optional*, only for *table* tag

```
name='cellspacing'  
abstraction='character'  
model=space between cells in pixels
```

### Cellpadding Property

This property specifies the space between an HTML table cell's border and its content.

*optional*, only for *table* tag

```
name='cellpadding'  
abstraction='character'  
model=space between cell border and content
```

### Border Property

This property specifies the width of the HTML table's border.

*optional*, only for *table* tag

```
name='border'  
abstraction='character'  
model=table border width
```

### **HRef Property**

This property specifies the reference (link) to be invoked when activating the HTML element.

*optional*, only for *a* tag

```
name='href'  
abstraction='character'  
model=an HTML reference
```



## 4 Logic Models

### 4.1 Bit Manipulation

#### 4.1.1 Shift

This operation shifts the bits within the number by the given positions.

#### Example

```
<part name="shift_bits" channel="inline" abstraction="operation" model="shift">
  <property name="number" channel="inline" abstraction="integer" model="8"/>
  <property name="direction" channel="inline" abstraction="character" model="right"/>
  <property name="position" channel="inline" abstraction="integer" model="2"/>
  <property name="result" channel="inline" abstraction="knowledge" model=".app.result"/>
</part>
```

#### Number Property

This property specifies the number whose bits are to be shifted.

*required*

name='number'

abstraction='integer' | 'knowledge' | 'encapsulated'

model=the actual number (or knowledge model)

## Direction Property

The direction in which to shift.

*required*

```
name='direction'
abstraction='character'
model='left' | 'right'
```

## Position Property

The number of positions by which the bits of the given number are to be shifted.

*required*

```
name='position'
abstraction='integer' | 'knowledge' | 'encapsulated'
model=number of positions to shift by
```

## Result Property

This is the result knowledge model in which to store the number whose bits were shifted.

*required*

```
name='result'
abstraction='knowledge' | 'encapsulated'
model=result knowledge model
```

### 4.1.2 Rotate

This operation rotates the bits within the number by the given positions.

#### Example

```
<part name="rotate_bits" channel="inline" abstraction="operation" model="rotate">
  <property name="number" channel="inline" abstraction="knowledge" model=".app.number"/>
  <property name="direction" channel="inline" abstraction="character" model="left"/>
```

```
<property name="position" channel="inline" abstraction="integer" model="1"/>
<property name="result" channel="inline" abstraction="knowledge" model=".app.result"/>
</part>
```

### Number Property

This property specifies the number whose bits are to be rotated.

*required*

```
name='number'
abstraction='integer' | 'knowledge' | 'encapsulated'
model=the actual number (or knowledge model)
```

### Direction Property

The direction in which to rotate.

*required*

```
name='direction'
abstraction='character'
model='left' | 'right'
```

### Position Property

The number of positions by which the bits of the given number are to be rotated.

*required*

```
name='position'
abstraction='integer' | 'knowledge' | 'encapsulated'
model=number of positions to rotate by
```

### Result Property

This is the result knowledge model in which to store the number whose bits were rotated.

*required*

```
name='result'
abstraction='knowledge' | 'encapsulated'
model=result knowledge model
```

### 4.1.3 Set Bit

This operation sets the bit at the given position within the given number. *Set* means setting the bit to *true*.

#### Example

```
<part name="set_some_bit" channel="inline" abstraction="operation" model="set_bit">
  <property name="number" channel="inline" abstraction="knowledge" model=".app.number"/>
  <property name="position" channel="inline" abstraction="knowledge" model=".app.pos"/>
  <property name="result" channel="inline" abstraction="knowledge" model=".app.result"/>
</part>
```

#### Number Property

This property specifies the number of which one bit is to be set.

*required*

```
name='number'
abstraction='integer' | 'knowledge' | 'encapsulated'
model=the actual number (or knowledge model)
```

#### Position Property

The position of the bit whose value is to be set (set to *true*).

*required*

```
name='position'
abstraction='integer' | 'knowledge' | 'encapsulated'
model=position of the bit
```

## Result Property

This is the result knowledge model in which to store the number of which one bit was set (to *true*).

*required*

```
name='result'  
abstraction='knowledge' | 'encapsulated'  
model=result knowledge model
```

### 4.1.4 Reset Bit

This operation resets the bit at the given position within the given number. *Reset* means setting the bit to *false*.

#### Example

```
<part name="reset_some_bit" channel="inline" abstraction="operation" model="reset_bit">  
  <property name="number" channel="inline" abstraction="integer" model="80"/>  
  <property name="position" channel="inline" abstraction="integer" model="2"/>  
  <property name="result" channel="inline" abstraction="knowledge" model=".app.result"/>  
</part>
```

## Number Property

This property specifies the number of which one bit is to be reset.

*required*

```
name='number'  
abstraction='integer' | 'knowledge' | 'encapsulated'  
model=the actual number (or knowledge model)
```

## Position Property

The position of the bit whose value is to be reset (set to *false*).

*required*

```
name='position'
abstraction='integer' | 'knowledge' | 'encapsulated'
model=position of the bit
```

## Result Property

This is the result knowledge model in which to store the number of which one bit was reset (to *false*).

*required*

```
name='result'
abstraction='knowledge' | 'encapsulated'
model=result knowledge model
```

### 4.1.5 Get Bit

This operation retrieves the bit at the given position, within the given number.

## Example

```
<part name="get_some_bit" channel="inline" abstraction="operation" model="get_bit">
  <property name="number" channel="inline" abstraction="integer" model="127"/>
  <property name="position" channel="inline" abstraction="integer" model="3"/>
  <property name="result" channel="inline" abstraction="knowledge" model=".app.result"/>
</part>
```

## Number Property

This property specifies the number of which one bit is to be retrieved.

*required*

```
name='number'
abstraction='integer' | 'knowledge' | 'encapsulated'
model=the actual number (or knowledge model)
```

## Position Property

The position of the bit whose value is to be retrieved.

*required*

```
name='position'  
abstraction='integer' | 'knowledge' | 'encapsulated'  
model=position of the bit
```

## Result Property

This is the result knowledge model in which to store the value of the bit that was retrieved from the given number. It may be either *true* or *false*.

*required*

```
name='result'  
abstraction='knowledge' | 'encapsulated'  
model=result knowledge model
```

## 4.2 Boolean Logic

### 4.2.1 NOT

This operation applies the logic NOT operator to the given boolean operand.

#### Example

```
<part name="apply_not" channel="inline" abstraction="operation" model="not">  
  <property name="operand" channel="inline" abstraction="boolean" model="true"/>  
  <property name="result" channel="inline" abstraction="knowledge" model=".app.result"/>  
</part>
```

## Operand Property

This is the operand of the boolean operation.

*required*

```
name='operand'
abstraction='boolean' | 'knowledge' | 'encapsulated'
model=boolean value or knowledge model
```

## Result Property

This is the result of the boolean operation. It may be either *true* or *false*.

*required*

```
name='result'
abstraction='knowledge' | 'encapsulated'
model=knowledge model
```

## 4.2.2 NEG

This operation applies the logic NEG operator to the given boolean operand.

### Example

```
<part name="apply_neg" channel="inline" abstraction="operation" model="neg">
  <property name="operand" channel="inline" abstraction="knowledge" model=".app.value"/>
  <property name="result" channel="inline" abstraction="knowledge" model=".app.result"/>
</part>
```

## Operand Property

This is the operand of the boolean operation.

*required*



```
name='operand'  
abstraction='boolean' | 'knowledge' | 'encapsulated'  
model=boolean value or knowledge model
```

### Result Property

This is the result of the boolean operation. It may be either *true* or *false*.

*required*

```
name='result'  
abstraction='knowledge' | 'encapsulated'  
model=knowledge model
```

### 4.2.3 AND

This operation applies the logic AND operator to the given boolean operands.

#### Example

```
<part name="apply_and" channel="inline" abstraction="operation" model="and">  
  <property name="operand_1" channel="inline" abstraction="knowledge" model=".app.value"/>  
  <property name="operand_2" channel="inline" abstraction="boolean" model="false"/>  
  <property name="result" channel="inline" abstraction="knowledge" model=".app.result"/>  
</part>
```

### Operand 1 Property

This is the first operand of the boolean operation.

*required*

```
name='operand_1'  
abstraction='boolean' | 'knowledge' | 'encapsulated'  
model=boolean value or knowledge model
```

### Operand 2 Property

This is the second operand of the boolean operation.

*required*

```
name='operand_2'
abstraction='boolean' | 'knowledge' | 'encapsulated'
model=boolean value or knowledge model
```

### Result Property

This is the result of the boolean operation. It may be either *true* or *false*.

*required*

```
name='result'
abstraction='knowledge' | 'encapsulated'
model=knowledge model
```

## 4.2.4 OR

This operation applies the logic OR operator to the given boolean operands.

### Example

```
<part name="apply_or" channel="inline" abstraction="operation" model="or">
  <property name="operand_1" channel="inline" abstraction="boolean" model="true"/>
  <property name="operand_2" channel="inline" abstraction="boolean" model="true"/>
  <property name="result" channel="inline" abstraction="knowledge" model=".app.result"/>
</part>
```

### Operand 1 Property

This is the first operand of the boolean operation.

*required*

```
name='operand_1'  
abstraction='boolean' | 'knowledge' | 'encapsulated'  
model=boolean value or knowledge model
```

## Operand 2 Property

This is the second operand of the boolean operation.

*required*

```
name='operand_2'  
abstraction='boolean' | 'knowledge' | 'encapsulated'  
model=boolean value or knowledge model
```

## Result Property

This is the result of the boolean operation. It may be either *true* or *false*.

*required*

```
name='result'  
abstraction='knowledge' | 'encapsulated'  
model=knowledge model
```

### 4.2.5 XOR

This operation applies the logic XOR operator to the given boolean operands.

#### Example

```
<part name="apply_xor" channel="inline" abstraction="operation" model="xor">  
  <property name="operand_1" channel="inline" abstraction="boolean" model="true"/>  
  <property name="operand_2" channel="inline" abstraction="boolean" model="false"/>  
  <property name="result" channel="inline" abstraction="knowledge" model=".app.result"/>  
</part>
```

### Operand 1 Property

This is the first operand of the boolean operation.

*required*

```
name='operand_1'  
abstraction='boolean' | 'knowledge' | 'encapsulated'  
model=boolean value or knowledge model
```

### Operand 2 Property

This is the second operand of the boolean operation.

*required*

```
name='operand_2'  
abstraction='boolean' | 'knowledge' | 'encapsulated'  
model=boolean value or knowledge model
```

### Result Property

This is the result of the boolean operation. It may be either *true* or *false*.

*required*

```
name='result'  
abstraction='knowledge' | 'encapsulated'  
model=knowledge model
```

## 4.3 Program Flow

### 4.3.1 Branch

This operation branches the program flow, depending on the given criterion flag. If the flag is true, another logic knowledge model may be executed than if the flag is false.

## Example

```
<part name="branch" channel="inline" abstraction="operation" model="branch">
  <property name="criterion" channel="inline" abstraction="knowledge" model=".app.crit"/>
  <property name="true" channel="inline" abstraction="knowledge" model=".app.add_address"/>
  <property name="false" channel="inline" abstraction="knowledge" model=".app.del_address"/>
</part>
```

## Criterion Property

This is the flag specifying which of the two models to execute.

*required*

```
name='criterion'
abstraction='boolean' | 'knowledge' | 'encapsulated'
model='true' | 'false' | knowledge model pointing to flag
```

## True Property

This is the logic knowledge model to be executed if the condition is true.

*required*

```
name='true'
abstraction='knowledge' | 'encapsulated'
model=knowledge model path if true
```

## False Property

This is the logic knowledge model to be executed if the condition is false.

*required*

```
name='false'
abstraction='knowledge' | 'encapsulated'
model=knowledge model path if false
```

### 4.3.2 Loop

This operation starts an endless loop that runs until the given break flag is set.

#### Example

```
<part name="loop_addresses" channel="inline" abstraction="operation" model="loop">
  <property name="model" channel="inline" abstraction="knowledge" model=".app.process"/>
  <property name="break" channel="inline" abstraction="knowledge" model=".app.break_flag"/>
</part>
```

#### Model Property

This is the knowledge model to be executed repeatedly by the loop.

*required*

```
name='model'
abstraction='knowledge' | 'encapsulated'
model=knowledge model
```

#### Break Property

This is the break flag. Once set, the loop will be left (exited). It may be either *true* or *false*.

*required*

```
name='break'
abstraction='knowledge' | 'encapsulated'
model=knowledge model
```

### 4.3.3 Count

This operation counts those parts of the given whole (compound), that match the given filter criteria.

## Example

```
<part name="count_addresses" channel="inline" abstraction="operation" model="count">
  <property name="compound" channel="inline" abstraction="encapsulated" model=".app.name"/>
  <property name="selection" channel="inline" abstraction="character" model="prefix"/>
  <property name="filter" channel="inline" abstraction="character" model="address"/>
  <property name="result" channel="inline" abstraction="knowledge" model=".app.result"/>
</part>
```

## Compound Property

The compound whose parts are to be counted.

*required*

```
name='compound'
abstraction='knowledge' | 'encapsulated'
model=compound knowledge model
```

## Selection Property

This property selects the kind of filter to be applied for counting the compound's parts.

*required*

```
name='selection'
abstraction='character'
model='full' | 'prefix' | 'suffix' | 'part'
```

## Filter Property

The filter to compare the compound parts' names with. Only those parts will be counted whose name (full, prefix, suffix, part) matches the filter string.

*required*

```
name='filter'
abstraction='character'
model=filter string
```

## Result Property

The knowledge model in which to store the result.

*required*

```
name='result'
abstraction='knowledge' | 'encapsulated'
model=knowledge model
```

### 4.3.4 Get

#### Example

```
<part name="get_fifth_address" channel="inline" abstraction="operation" model="get">
  <property name="compound" channel="inline" abstraction="knowledge" model=".app.adr"/>
  <property name="index" channel="inline" abstraction="integer" model="4"/>
  <property name="description" channel="inline" abstraction="character" model="name"/>
  <property name="result" channel="inline" abstraction="knowledge" model=".app.result"/>
</part>
```

## Compound Property

This is the compound whose element is to be retrieved.

*required*

```
name='compound'
abstraction='knowledge' | 'encapsulated'
model='integer' | 'character'
```

## Index Property

This is the index of the element to be retrieved, within the compound.

*required*



```
name='index'  
abstraction='integer'  
model=position of element
```

### Description Property

This property determines which data of the compound's part to retrieve.

*required*

```
name='description'  
abstraction='character'  
model='name' | 'abstraction'
```

### Result Property

This is the result knowledge model in which to store the retrieved element.

*required*

```
name='result'  
abstraction='knowledge' | 'encapsulated'  
model=result knowledge model
```

## 4.4 Comparison

### 4.4.1 Compare

This operation compares two given operands.

#### Example

```
<part name="compare_loop_index" channel="inline" abstraction="operation" model="compare">  
  <property name="comparison" channel="inline" abstraction="character" model="greater_or_equal"/>  
  <property name="left_side" channel="inline" abstraction="knowledge" model=".app.tmp.loop_index"/>  
  <property name="right_side" channel="inline" abstraction="integer" model="5"/>  
  <property name="result" channel="inline" abstraction="knowledge" model=".app.tmp.break_flag"/>
```

</part>

### Comparison Property

This is the kind of comparison to be applied.

*required*

```
name='comparison'  
abstraction='character'  
model='equal' | 'smaller' | 'greater' | 'smaller_or_equal' | 'greater_or_equal'
```

### Left Side Property

This is the left side value of the comparison.

*required*

```
name='left_side'  
abstraction='boolean' | 'integer' | 'float' | 'character' | 'knowledge' | 'encapsulated'  
model=value or knowledge model
```

### Right Side Property

This is the right side value of the comparison.

*required*

```
name='right_side'  
abstraction='boolean' | 'integer' | 'float' | 'character' | 'knowledge' | 'encapsulated'  
model=value or knowledge model
```

### Result Property

This is the knowledge model in which the comparison result is stored.

*required*

```

name='result'
abstraction='boolean' | 'knowledge' | 'encapsulated'
model=result knowledge model

```

### Selection Property

This property selects which part of two string values shall be compared.

*optional*, only if comparing values of abstraction "character"

```

name='selection'
abstraction='character'
model='full' | 'prefix' | 'suffix' | 'part'

```

## 4.5 Arithmetic

### 4.5.1 Add

This operation is a simple addition of two numbers or strings.

#### Example

```

<part name="add_numbers" channel="inline" abstraction="operation" model="add">
  <property name="summand_1" channel="inline" abstraction="knowledge" model=".app.summand_1"/>
  <property name="summand_2" channel="inline" abstraction="knowledge" model=".app.summand_2"/>
  <property name="sum" channel="inline" abstraction="knowledge" model=".app.sum"/>
</part>

```

### Abstraction Property

This property specifies the abstraction of the properties `summand_1`, `summand_2` and `sum`. In other words, the models of these properties have to have the same abstraction.

*required*

```
name='abstraction'
abstraction='character'
model='integer' | 'character'
```

### Summand 1 Property

The first summand for the addition.

*required*

```
name='summand_1'
abstraction='integer' | 'character' | 'knowledge' | 'encapsulated'
model=number or knowledge model path
```

### Summand 2 Property

The second summand for the addition.

*required*

```
name='summand_2'
abstraction='integer' | 'character' | 'knowledge' | 'encapsulated'
model=number or knowledge model path
```

### Sum Property

The sum resulting from the addition.

*required*

```
name='sum'
abstraction='integer' | 'character' | 'knowledge' | 'encapsulated'
model=knowledge model path
```

## 4.5.2 Subtract

This operation subtracts one number from another which results in their difference.

## Example

```
<part name="subtract_numbers" channel="inline" abstraction="operation" model="subtract">
  <property name="minuend" channel="inline" abstraction="integer" model="10"/>
  <property name="subtrahend" channel="inline" abstraction="integer" model="7"/>
  <property name="difference" channel="inline" abstraction="knowledge" model=".app.difference"/>
</part>
```

## Minuend Property

This is the minuend, i.e. the number to be subtracted from.

*required*

```
name='minuend'
abstraction='integer' | 'knowledge' | 'encapsulated'
model=number or knowledge model path
```

## Subtrahend Property

This is the subtrahend, i.e. the number to be subtracted.

*required*

```
name='subtrahend'
abstraction='integer' | 'knowledge' | 'encapsulated'
model=number or knowledge model path
```

## Difference Property

This is the difference between minuend and subtrahend.

*required*

```
name='difference'
abstraction='integer' | 'knowledge' | 'encapsulated'
model=knowledge model path
```

### 4.5.3 Multiply

This operation multiplies two numbers which results in the product.

#### Example

```
<part name="multiply_numbers" channel="inline" abstraction="operation" model="multiply">
  <property name="factor_1" channel="inline" abstraction="integer" model="2"/>
  <property name="factor_2" channel="inline" abstraction="knowledge" model=".app.factor"/>
  <property name="product" channel="inline" abstraction="knowledge" model=".app.product"/>
</part>
```

#### Factor 1 Property

This is the first factor of the multiplication.

*required*

```
name='factor_1'
abstraction='integer'
model=number or knowledge model
```

#### Factor 2 Property

This is the second factor of the multiplication.

*required*

```
name='factor_2'
abstraction='integer'
model=number or knowledge model
```

#### Product Property

This is the product as result of the multiplication.

*required*

```
name='product'  
abstraction='integer'  
model=number or knowledge model
```

#### 4.5.4 Divide

This operation divides one number by another, which results in the quotient. If the dividend is not a multiple of the divisor, then the operation will return the remainder as additional property.

##### Example

```
<part name="divide_numbers" channel="inline" abstraction="operation" model="divide">  
  <property name="dividend" channel="inline" abstraction="integer" model="table"/>  
  <property name="divisor" channel="inline" abstraction="integer" model="divisor"/>  
  <property name="quotient" channel="inline" abstraction="integer" model="quotient"/>  
  <property name="remainder" channel="inline" abstraction="integer" model="remainder"/>  
</part>
```

##### Dividend Property

This is the dividend, i.e. the number to be divided.

*required*

```
name='dividend'  
abstraction='integer' | 'knowledge' | 'encapsulated' model=number or knowledge model
```

##### Divisor Property

This is the divisor, i.e. the number to be divided by.

*required*

```
name='divisor'  
abstraction='integer' | 'knowledge' | 'encapsulated' model=number or knowledge model
```

## Quotient Property

This is the quotient, i.e. the number resulting from the division.

*required*

name='quotient'

abstraction='integer' | 'knowledge' | 'encapsulated' model=number or knowledge model

## Remainder Property

This is the remainder, i.e. the difference between dividend and quotient.

*required*

name='remainder'

abstraction='integer' | 'knowledge' | 'encapsulated' model=number or knowledge model

## 4.6 Memory Management

### 4.6.1 Create

This operation creates a new knowledge model in memory.

#### Example

```
<part name="create_addresses" channel="inline" abstraction="operation" model="create">
  <property name="name" channel="inline" abstraction="character" model="addresses"/>
  <property name="abstraction" channel="inline" abstraction="character" model="compound"/>
  <property name="element" channel="inline" abstraction="character" model="part"/>
  <property name="compound" channel="inline" abstraction="knowledge" model=".app"/>
</part>
```

#### Name Property

This is the name of the knowledge model to be created.



*required*

```
name='name'  
abstraction='character'  
model=knowledge model name
```

### **Abstraction Property**

This is the abstraction (type) of the knowledge model to be created.

*required*

```
name='abstraction'  
abstraction='character'  
model='boolean' | 'integer' | 'float' | 'character' | 'compound'
```

### **Element Property**

This property decides about the kind of element (knowledge model) to be created. A part element will be added to the model's part hierarchy; a meta element will be added to the model's details hierarchy.

*required*

```
name='element'  
abstraction='character'  
model='part' | 'meta'
```

### **Compound Property**

This property specifies the compound knowledge model to which to add to the new part/meta knowledge model.

*required*

```
name='compound'  
abstraction='knowledge' | 'encapsulated'  
model=whole knowledge model path
```

## 4.6.2 Destroy

This operation destroys the given knowledge model and releases the memory that was occupied by it.

### Example

```
<part name="destroy_addresses" channel="inline" abstraction="operation" model="destroy">
  <property name="model" channel="inline" abstraction="knowledge" model=".app.addresses"/>
</part>
```

### Model Property

This is the knowledge model to be destroyed.

*required*

```
name='model'
abstraction='knowledge' | 'encapsulated'
model='knowledge model'
```

## 4.6.3 Copy

This operation copies a value with the given abstraction from the source- to the destination model.

### Example

```
<part name="copy_title" channel="inline" abstraction="operation" model="copy">
  <property name="abstraction" channel="inline" abstraction="character" model="character"/>
  <property name="source" channel="inline" abstraction="character" model="My Book"/>
  <property name="destination" channel="inline" abstraction="knowledge" model=".app.title"/>
</part>
```

## Abstraction Property

This is the abstraction of the values. CAUTION! It has to be identical for both, source- and destination value!

*required*

```
name='abstraction'  
abstraction='character'  
model='boolean' | 'integer' | 'character'
```

## Source Property

This is the source value.

*required*

```
name='source'  
abstraction='boolean' | 'integer' | 'character' | 'knowledge' | 'encapsulated'  
model=value or knowledge model
```

## Destination Property

This is the destination value.

*required*

```
name='destination'  
abstraction='boolean' | 'integer' | 'character' | 'knowledge' | 'encapsulated'  
model=value or knowledge model
```

### 4.6.4 Move

This operation moves the part knowledge model to a new whole (compound) knowledge model.

## Example

```
<part name="move_addresses" channel="inline" abstraction="operation" model="move">
  <property name="part" channel="inline" abstraction="knowledge" model=".app.addresses"/>
  <property name="whole" channel="inline" abstraction="knowledge" model=".app.domain"/>
</part>
```

## Part Property

This is the part knowledge model that is to be moved.

*required*

```
name='part'
abstraction='knowledge' | 'encapsulated'
model=part knowledge model
```

## Whole Property

This is the whole knowledge model to which to add to the part knowledge model.

*required*

```
name='whole'
abstraction='knowledge' | 'encapsulated'
model=whole knowledge model
```

## 4.7 Lifecycle Management

### 4.7.1 Startup

This operation starts up the given service.

## Example

```
<part name="startup_wui" channel="inline" abstraction="operation" model="startup">
  <property name="service" channel="inline" abstraction="character" model="www"/>
</part>
```

```
<property name="namespace" channel="inline" abstraction="character" model="inet6"/>
<property name="style" channel="inline" abstraction="character" model="stream"/>
<property name="address" channel="inline" abstraction="character" model="any"/>
</part>
```

## Service Property

This is the service to be started up.

*required*

```
name='service'
abstraction='character'
model='signal' | 'shell' | 'standard_output' | 'gnu_linux_console' | 'x_window_system'
| 'www' | 'cyboi'
```

## Namespace Property

The namespace of the socket.

*optional*, only if service is *www* or *cyboi*

```
name='namespace'
abstraction='character'
model='local' | 'inet' | 'inet6' | 'ns' | 'iso' | 'ccitt' | 'implink' | 'route'
```

## Style Property

The style of socket communication.

*optional*, only if service is *www* or *cyboi*

```
name='style'
abstraction='character'
model='stream' | 'datagram' | 'raw'
```

## Address Property

This is the address of hosts communicating with this system via socket.

*optional*, only if service is *www* or *cyboi*

name='address'

abstraction='character'

model='loopback' | 'any'

### 4.7.2 Shutdown

This operation shuts down the given service.

#### Example

```
<part name="shutdown_gui" channel="inline" abstraction="operation" model="shutdown">
  <property name="service" channel="inline" abstraction="character" model="x_window_system"/>
</part>
```

## Service Property

This is the service to be shut down.

*required*

name='service'

abstraction='character'

model='signal' | 'shell' | 'standard\_output' | 'gnu\_linux\_console' | 'x\_window\_system'  
| 'www' | 'cyboi'

### 4.7.3 Exit

This operation initiates the shutdown sequence for the application system.

## Example

```
<part name="exit_system" channel="inline" abstraction="operation" model="exit"/>
```

## 4.8 Communication

### 4.8.1 Send

This operation is able to send a message via textual, graphical or web user interface, or to the file system or also as shell output directly.

## Example

```
<part name="send_menu" channel="inline" abstraction="operation" model="send">
  <property name="channel" channel="inline" abstraction="character" model="gnu_linux_console"/>
  <property name="language" channel="inline" abstraction="character" model="tui"/>
  <property name="message" channel="inline" abstraction="knowledge" model=".app.tui"/>
  <property name="area" channel="inline" abstraction="knowledge" model=".app.tui.menu"/>
  <property name="clean" channel="inline" abstraction="boolean" model="true"/>
</part>
```

## Channel Property

The channel via which to send the message.

*required*

```
name='channel'
abstraction='character'
model='inline' | 'file' | 'standard_output' | 'gnu_linux_console' | 'x_window_system'
| 'http'
```

## Language Property

The language into which to encode the message before sending it.

*required*

```
name='language'  
abstraction='character'  
model='tui' | 'gui' | 'wui'
```

### **Mode Property**

The mode of communication.

*optional*, only if channel is *http*

```
name='mode'  
abstraction='character'  
model='client' | 'server'
```

### **Namespace Property**

The namespace of the socket.

*optional*, only if channel is *http*

```
name='namespace'  
abstraction='character'  
model='local' | 'inet' | 'inet6' | 'ns' | 'iso' | 'ccitt' | 'implink' | 'route'
```

### **Style Property**

The style of communication.

*optional*, only if channel is *http*

```
name='style'  
abstraction='character'  
model='stream' | 'datagram' | 'raw'
```

### **Receiver Property**

The name of the system receiving the message.



*required*

```
name='receiver'  
abstraction='character'  
model=name of receiving system
```

### **Message Property**

The actual message to be sent to another system.

*required*

```
name='message'  
abstraction='knowledge' | 'encapsulated'  
model=message knowledge model path
```

### **Area Property**

The user interface area to be repainted. It is normally just a part of the whole user interface model. This property helps to speed up repainting while avoiding user interface flickering.

*optional*

```
name='area'  
abstraction='knowledge' | 'encapsulated'  
model=knowledge path to part model to be repainted
```

### **Clean Property**

This property indicates whether or not to clear the screen before painting a user interface.

*optional*

```
name='clean'  
abstraction='boolean'  
model='true' | 'false'
```

## New Line Property

This property indicates whether or not to add a new line after having printed the message on screen.

*optional*

```
name='new_line'
abstraction='boolean'
model='true' | 'false'
```

## 4.8.2 Receive

This operation receives data from the given data source.

### Example

```
<part name="receive_patients" channel="inline" abstraction="operation" model="receive">
  <property name="channel" channel="inline" abstraction="character" model="file"/>
  <property name="language" channel="inline" abstraction="character" model="xdt"/>
  <property name="message" channel="inline" abstraction="character" model="import/1.bde"/>
  <property name="model" channel="inline" abstraction="knowledge" model=".app.xdt"/>
</part>
```

## Channel Property

The channel via which to receive the message.

*required*

```
name='channel'
abstraction='character'
model='inline' | 'file'
```

## Language Property

This is the language (abstraction, type, structure) of the data received.

*required*

```
name='language'
abstraction='character'
model= 'boolean' | 'character' | 'wide_character' | 'integer' | 'unsigned_long' | 'double'
| 'fraction' | 'complex' | 'date_time' | 'yyyy-mm-dd_date_time' | 'cybol' | 'tui' | 'gui'
| 'wui' | 'ogg' | 'mp3' | 'jpeg' | 'png' | 'gif' | 'bmp' | 'cybop_model_diagram' | 'xdt' |
'hxp' | 'latex' | 'rtf' | 'sgml' | 'tex' | 'xhtml' | 'mpeg' | 'avi' | 'qt' | 'tar' | 'tgz'
| 'zip' | 'rar' | 'kwd' | 'odt' | 'sxw' | 'http' | 'https' | 'ftp'
```

### Message Property

This is the source (knowledge template) from where to receive data.

*required*

```
name='message'
abstraction='character'
model=path to a file
```

### Model Property

This is the compound model to be filled with the data received.

*required*

```
name='model'
abstraction='character'
model=knowledge model path
```

### Details Property

This is the compound details to be filled with the data received.

*required*

```
name='details'
abstraction='character'
model=knowledge model path
```

### Root Property

This property specifies the knowledge model that will serve as the root.

*required*

name='root'

abstraction='knowledge' | 'encapsulated' model=root model knowledge path

### Style Property

This is the style of socket communication.

*required*

name='style'

abstraction='knowledge' | 'encapsulated' model='stream' | 'datagram' | 'raw'

### Commands Property

This property specifies the knowledge model containing the commands that the user interface should react to.

*optional*, only if a user interface thread is to react to certain commands

name='commands'

abstraction='knowledge' | 'encapsulated' model=commands model knowledge path

### Blocking Property

This property specifies whether the receive process should be blocking. If it is, then application signals will not be processed while the receive operation is waiting for some message to arrive. Only if a message is actually received, the application will process it in form of a signal and then continue to wait.

*optional*

name='blocking'

abstraction='boolean' model='true' | 'false'

### 4.8.3 Interrupt

This operation interrupts a running service. If the given service is not running, the operation will do nothing.

#### Example

```
<part name="interrupt_console" channel="inline" abstraction="operation" model="interrupt">
  <property name="service" channel="inline" abstraction="character" model="gnu_linux_console"/>
</part>
```

#### Service Property

The service to be interrupted.

*required*

```
name='service'
abstraction='character'
model='signal' | 'shell' | 'standard_output' | 'gnu_linux_console' | 'x_window_system'
| 'www' | 'cyboi'
```

## 4.9 Shell Commands

### 4.9.1 Archive File

This operation archives the given files or directories. Internally, it just uses the corresponding shell command functionality.

#### Example

```
<part name="archive" channel="inline" abstraction="operation" model="archive_file">
  <property name="source" channel="inline" abstraction="character" model="*.dbf"/>
  <property name="create" channel="inline" abstraction="boolean" model="true"/>
  <property name="update" channel="inline" abstraction="boolean" model="true"/>
  <property name="bzip2" channel="inline" abstraction="boolean" model="false"/>
```

</part>

### Source Property

The path or shell expression pattern determining the directories and files to be compressed and/ or archived.

*required*

```
name='source'  
abstraction='character'  
model=directories and files to be compressed
```

### Create Property

This is the flag specifying whether or not to create the archive file, if it does not exist yet.

*required*

```
name='create'  
abstraction='boolean'  
model='true' | 'false'
```

### Update Property

This property specifies whether or not to update the archive file, if it already exists.

*required*

```
name='update'  
abstraction='boolean'  
model='true' | 'false'
```

### Bzip2 Property

This property specifies whether or not to use bzip2 (bz2) compression instead of the standard gzip (gz) compression.

*required*

```
name='bzip2'  
abstraction='boolean'  
model='true' | 'false'
```

## 4.9.2 Copy File

This operation copies the given files or directories. Internally, it just uses the corresponding shell command functionality.

### Example

```
<part name="copy_directory" channel="inline" abstraction="operation" model="copy">  
  <property name="recursive" channel="inline" abstraction="boolean" model="true"/>  
  <property name="source" channel="inline" abstraction="character" model="/home/cybop/src"/>  
  <property name="destination" channel="inline" abstraction="character" model="/home/backup"/>  
</part>
```

### Recursive Property

This is the flag specifying whether or not to copy recursively.

*required*

```
name='recursive'  
abstraction='boolean'  
model='true' | 'false'
```

### Source Property

This property specifies the source file or -directory.

*required*

```
name='source'  
abstraction='character'  
model=source file or directory
```

## Destination Property

This property specifies the destination file or -directory.

*required*

```
name='destination'  
abstraction='character'  
model=destination file or directory
```

### 4.9.3 Execute File

This operation executes the given file.

#### Example

```
<part name="execute_cyboi" channel="inline" abstraction="operation" model="execute">  
  <property name="file" channel="inline" abstraction="character" model="/home/cyboi/bin/cyboi"/>  
</part>
```

## File Property

This is the file to be executed.

*required*

```
name='file'  
abstraction='character'  
model=executable file
```

### 4.9.4 List File

This operation lists the file content of the given directory. Internally, it just uses the corresponding shell command functionality.



## Example

```
<part name="list_contents" channel="inline" abstraction="operation" model="list_file">
  <property name="directory" channel="inline" abstraction="character" model="/home/cybop"/>
  <property name="all" channel="inline" abstraction="boolean" model="true"/>
  <property name="long_listing" channel="inline" abstraction="boolean" model="true"/>
</part>
```

## Directory Property

This is the flag specifying whether or not to list the entire directory contents.

*required*

```
name='all'
abstraction='boolean'
model='true' | 'false'
```

## All Property

This is the flag specifying whether or not to list the entire directory contents.

*required*

```
name='all'
abstraction='boolean'
model='true' | 'false'
```

## Long Listing Property

This property specifies whether or not to display the contents in form of a long listing, which also shows access rights etc.

*required*

```
name='long_listing'
abstraction='boolean'
model='true' | 'false'
```



## 5 Examples

The following examples demonstrate how CYBOL's constructs may be used in practice. Also, attention is paid to how control structures of classical programming languages may be implemented in CYBOL. Furthermore, this section discusses how inheritance, containers and software patterns were considered in the design of CYBOL.

### 5.1 State Examples

The creation of composed state models is quite straightforward and clear, as the following CYBOL knowledge templates show.

#### 5.1.1 Model-Part Relation

The DocBook DTD [16] is one of many well-known specifications for structuring documents. The Linux Documentation Project (TLDP) [10] makes heavy use of it. DocBook is based on numerous XML tags with defined meaning. The following example shows how parts of a *Text Document* can be modelled differently, with at most four tags, using CYBOL:

```
<model>
  <part name="title" channel="inline" abstraction="string" model="Quo Vadis"/>
  <part name="author" channel="inline" abstraction="string" model="Henryk Sienkiewicz"/>
  <part name="date" channel="inline" abstraction="date" model="1896-01-01"/>
  <part name="contents" channel="file" abstraction="cybol" model="contents.cybol"/>
  <part name="chapter_$1" channel="file" abstraction="cybol" model="chapter_1.cybol"/>
  <part name="chapter_$2" channel="file" abstraction="cybol" model="chapter_2.cybol"/>
  <part name="chapter_$3" channel="file" abstraction="cybol" model="chapter_3.cybol"/>
  <part name="appendix" channel="file" abstraction="cybol" model="appendix.cybol"/>
</model>
```

### 5.1.2 Meta Properties

When modelling *Graphical User Interfaces* (GUI), a speciality to take care about is the *Position* of GUI elements within their surrounding container. GUI components may have very different orderings and positions. The *Java Swing* framework [3], for example, offers *BorderLayout*, *BoxLayout*, *CardLayout*, *FlowLayout*, *GridBagLayout* etc.

The following example of a GUI *Dialogue* assumes that an interpreter knows how to handle *Compass* layouts, which are the pendant of the above-mentioned *BorderLayout*:

```
<model>
  <part name="title" channel="inline" abstraction="string" model="Prescription Dialogue"/>
  <part name="menu_bar" channel="file" abstraction="cybol" model="menu_bar.cybol">
    <property name="position" channel="inline" abstraction="string" model="north"/>
  </part>
  <part name="tool_bar" channel="file" abstraction="cybol" model="tool_bar.cybol">
    <property name="position" channel="inline" abstraction="string" model="west"/>
  </part>
  <part name="contents_panel" channel="file" abstraction="cybol" model="contents_panel.cybol">
    <property name="position" channel="inline" abstraction="string" model="centre"/>
  </part>
  <part name="status_bar" channel="file" abstraction="cybol" model="status_bar.cybol">
    <property name="position" channel="inline" abstraction="string" model="south"/>
  </part>
</model>
```

Further meta information such as the *Colour* or *Size* of a GUI component may be given. The following example shows how a GUI *Button* may be modelled as part of some GUI panel. Again, properties like *size* are not modelled as part, because the button does not *consist* of them, in a structural way of thinking:

```
<model>
  <part name="exit_button" channel="file" abstraction="cybol" model="exit_button.cybol">
    <property name="position" channel="inline" abstraction="integer" model="0"/>
    <property name="size" channel="inline" abstraction="vector" model="80,20,1"/>
    <property name="colour" channel="inline" abstraction="rgb" model="127,127,127"/>
    <property name="action" channel="inline" abstraction="string" model="exit.cybol"/>
  </part>
</model>
```

### 5.1.3 External Resources

A *Text Document* like the one shown in the example above often contains graphical illustrations called *Figures*, which it may include from external files. One common graphics format is *Encapsulated PostScript* (EPS), for example. *Graphical User Interfaces* (GUI) as modelled before do contain *Icons*; a GUI button may contain a *Glyph* and so forth. CYBOL therefore offers ways for linking external resources, given in various formats, as shown in the following hypothetical knowledge template:

```
<model>
  <part name="pdf_document" channel="file" abstraction="pdf" model="example.pdf"/>
  <part name="ogg_audio" channel="file" abstraction="ogg" model="example.ogg"/>
  <part name="mpeg_video" channel="file" abstraction="mpeg" model="example.mpeg"/>
  <part name="eps_image" channel="file" abstraction="eps" model="example.eps"/>
  <part name="jpeg_image" channel="file" abstraction="jpeg" model="host.domain.tld/example.jpeg"/>
</model>
```

### 5.1.4 Serialised Model

A possible (but not necessarily recommended) alternative to the linking of external resources is to store such information (as binary code) *inline* in the CYBOL knowledge template. One case in which it is necessary to store all information inline in the model is *Serialisation*.

A CYBOL address management application that does not rely on the existence of a *Database Management System* (DBMS) probably has to store addresses in form of serialised files, such as the one shown following. It contains two parts representing dynamically extensible lists, one for *phone\_numbers* and another one for *addresses*:

```
<model>
  <part name="honorific_prefix" channel="inline" abstraction="string" model="Dr."/>
  <part name="given_name" channel="inline" abstraction="string" model="Tux"/>
  <part name="family_name" channel="inline" abstraction="string" model="Penguin"/>
  <part name="phone_numbers" channel="inline" abstraction="cybol" model="(
    <part name="home" channel="inline" abstraction="string" model="123"/>
    <part name="work" channel="inline" abstraction="string" model="456"/>
    <part name="mobile" channel="inline" abstraction="string" model="789"/>
  )"/>
  <part name="addresses" channel="inline" abstraction="cybol" model="(
    ...
  )"/>
</model>
```

The serialisation of CYBOL models causes one problem: Due to the double hierarchy to which belong *Whole-Part* relations (stored in XML attributes) and *Meta Information* (stored in XML tags), it is not possible to store CYBOL models in an XML-conform manner. Instead of referencing external files containing the corresponding CYBOL *Part* models, a serialised *Whole* model has to contain these inline.

While XML tags were invented as pairs consisting of a *begin* and an *end* tag, XML attribute values are enclosed by simple quotation marks. Hence, the beginning markup of an attribute value does not look any different than its ending markup. This is a true problem, because serialised whole-part hierarchies of CYBOL models, with attribute values containing complete sub models with their own attributes, would get completely mixed up in pure XML notation.

It was therefore inevitable to break XML-conformity and introduce two additional markup tokens "( and )", indicating the beginning and end of an XML attribute value. The tokens are extensions of the quotation marks of standard XML attributes, with one left/ right parenthesis, respectively. That way, the degree to which attributes are nested becomes countable and it is always clear to which tag an attribute belongs.

### 5.1.5 Meta Constraints

The example of this section shows a possible *Debian GNU/Linux* [14] *Package* definition, written in CYBOL:

```
<model>
  <part name="name" channel="inline" abstraction="string" model="resmedicinae"/>
  <part name="version" channel="inline" abstraction="string" model="0.1.0.0"/>
  <part name="section" channel="inline" abstraction="string" model="science"/>
  <part name="priority" channel="inline" abstraction="string" model="optional"/>
  <part name="architecture" channel="inline" abstraction="string" model="all"/>
  <part name="packages" channel="file" abstraction="cybol" model="resmedicinae-packages"/>
  <part name="files" channel="file" abstraction="cybol" model="resmedicinae-files"/>
  <part name="maintainer" channel="inline" abstraction="string" model="Happy Coder"/>
  <part name="description" channel="inline" abstraction="string" model="Medical System"/>
</model>
```

The part called *packages* in the example above references an external CYBOL knowledge template, which is displayed below. It represents a list of packages having different versions

and varying strengths of dependency. The *strength* property of the last of these packages has the model value *suggests* and, it contains meta information about that property, namely a *constraint*. Constraints can be, for example: minima, maxima or a choice of possible values, as in this case.

```
<model>
  <part name="cyboi" channel="inline" abstraction="string" model="cyboi">
    <property name="strength" channel="inline" abstraction="string" model="depends"/>
    <property name="version" channel="inline" abstraction="string" model=">= 1.0.0.0"/>
    <property name="conflicts" channel="inline" abstraction="string" model="< 1.0.0.0"/>
  </part>
  <part name="cybol-healthcare" channel="inline" abstraction="string" model="cybol-healthcare">
    <property name="strength" channel="inline" abstraction="string" model="depends"/>
    <property name="version" channel="inline" abstraction="string" model=">= 0.1.0.0"/>
  </part>
  <part name="resadmin" channel="inline" abstraction="string" model="resadmin">
    <property name="strength" channel="inline" abstraction="string" model="recommends"/>
    <property name="version" channel="inline" abstraction="string" model=">= 0.8.0.0"/>
  </part>
  <part name="resmedicinae-doc" channel="inline" abstraction="string" model="resmedicinae-doc">
    <property name="strength" channel="inline" abstraction="string" model="suggests">
      <constraint name="choice" channel="inline" abstraction="set" model="suggests,recommends"/>
    </property>
    <property name="version" channel="inline" abstraction="string" model=">= 0.1.0.0"/>
  </part>
</model>
```

## 5.2 Logic Examples

The CYBOL implementation of logic models needs more detailed explanation, in particular the use of special control structures as known from *Structured and Procedural Programming* (SPP).

### 5.2.1 Operation Call

As stated previously, logic models may access and manipulate state models. The simplest form of a logic model is an operation with associated input/ output (i/o) state models. The following CYBOL knowledge template calls an *add* operation, handing over i/o parameters as *properties* of the corresponding *part*:

```

<model>
  <part name="addition" channel="inline" abstraction="operation" model="add">
    <property name="abstraction" channel="inline" abstraction="character" model="integer"/>
    <property name="summand_1" channel="inline" abstraction="integer" model="1"/>
    <property name="summand_2" channel="inline" abstraction="knowledge" model=".app.summand"/>
    <property name="sum" channel="inline" abstraction="knowledge" model=".app.result"/>
  </part>
</model>

```

The example nicely shows how state models can be given in various formats. The *summand\_1* is given as constant value, defined directly in the knowledge template. Its type of abstraction is *integer*. The *summand\_2*- and *sum* parameters, on the other hand, are given as dot-separated references to the runtime tree of knowledge models. Their type of abstraction is therefore *knowledge*.

## 5.2.2 Algorithm Division

Compound logic models like *Algorithms*, which SPP languages implement using nested *Blocks*, can be expressed in CYBOL as well. It does not provide blocks in the classical sense, but its hierarchical structure allows to subdivide compound knowledge templates, and to cascade compound logic as well as primitive operations. The following example calls an addition operation, before a compound algorithm, situated in an external CYBOL file, gets executed:

```

<model>
  <part name="addition" channel="inline" abstraction="operation" model="add">
    <property name="summand_1" channel="inline" abstraction="knowledge" model="domain.number_1"/>
    <property name="summand_2" channel="inline" abstraction="knowledge" model="domain.number_2"/>
    <property name="sum" channel="inline" abstraction="knowledge" model="domain.number_3"/>
  </part>
  <part name="algorithm" channel="file" abstraction="cybol" model="logic/algorithm.cybol"/>
</model>

```

## 5.2.3 Simple Assignment

CYBOL does not know *Variables* as used in classical languages. All states a system may take on are represented by just one *Knowledge Tree*, which applications may access in a defined manner (dot-separated knowledge paths). Consequently, *Assignments* are done differently



in CYBOL than in classical programming languages. All kinds of state changes go back to a manipulation of the one knowledge tree:

```
<model>
  <part name="copy_value" channel="inline" abstraction="operation" model="copy">
    <property name="source" channel="inline" abstraction="knowledge" model="domain.name"/>
    <property name="destination" channel="inline" abstraction="knowledge" model="gui.name"/>
  </part>
  <part name="move_branch" channel="inline" abstraction="operation" model="move">
    <property name="source" channel="inline" abstraction="knowledge" model="address_1.phone"/>
    <property name="destination" channel="inline" abstraction="knowledge" model="address_2"/>
  </part>
</model>
```

The first operation in the example above copies a value between two branches of the tree. Only primitive values can be copied. The second operation removes a whole tree branch (referenced by the *source* property) from one parent node, and adds it to another (referenced by the *destination* property).

## 5.2.4 Loop as Operation

*Looping* is a major technique for the effective processing of whole stacks of data. As many other control structures, it is simplified to a logic operation, in CYBOL.

The *loop* operation needs two parameters to be functional: a *break* flag as means of interruption and a logic *model* to be executed in each loop cycle (figure 5.1). An *index* counting loop cycles is not given, as it is in the responsibility of the logic *model* to manage that index, just like the setting of the *break* flag, internally. The following example dynamically creates a table consisting of a number of rows:

```
<model>
  <part name="creat_table_body" channel="inline" abstraction="operation" model="loop">
    <property name="break" channel="inline" abstraction="knowledge" model=".domain.flag"/>
    <property name="model" channel="inline" abstraction="knowledge" model=".logic.create_rows"/>
  </part>
</model>
```

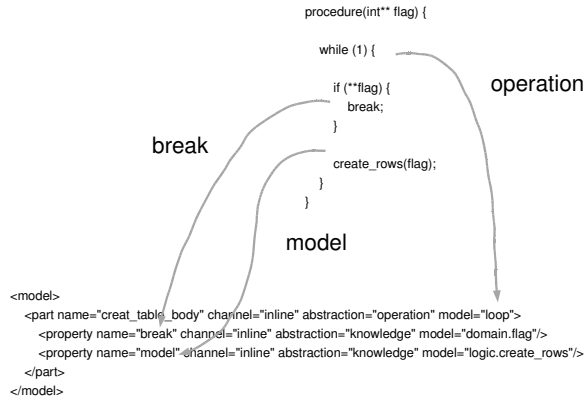


Figure 5.1: Loop Control Structure and Elements in C and CYBOL

### 5.2.5 Conditional Execution

An obviously presupposed part in the previous example is a logic setting the break *condition* (flag). If the break flag was not set, the loop would run endlessly. The following knowledge template therefore shows a *comparison* operation, as it could stand at the end of the loop's logic model, referenced by the *model* property in the previous example. After having compared the current loop index with a maximum loop count number, the break flag may or may not be set. When entering its next cycle, the loop operation checks whether the flag is set. If so, the loop is stopped:

```

<model>
  <part name="comparison" channel="inline" abstraction="operation" model="compare">
    <property name="operator" channel="inline" abstraction="character" model="greater_or_equal"/>
    <property name="left_side" channel="inline" abstraction="knowledge" model=".domain.index"/>
    <property name="right_side" channel="inline" abstraction="knowledge" model=".domain.count"/>
    <property name="result" channel="inline" abstraction="knowledge" model=".domain.flag"/>
  </part>
</model>

```

Flags as one of the earliest techniques used in computing (in software as well as in hardware) are the perfect means for controlling the execution of primitive logic models, namely operations. They represent a condition set as result of another logic model – the latter often

```

procedure(int** index, int** count) {
    if (**index >= **count) {
        true_model();
    } else {
        false_model();
    }
}

procedure(int** index, int** count, int** flag) {
    if (**index >= **count) {
        **flag = 1;
    }
    if (**flag) {
        true_model();
    } else {
        false_model();
    }
}

```

```

<model>
<part name="comparison" channel="inline" abstraction="operation" model="compare">
<property name="operand_1" channel="inline" abstraction="knowledge" model="domain.index"/>
<property name="operand_2" channel="inline" abstraction="knowledge" model="domain.count"/>
<property name="operator" channel="inline" abstraction="string" model="greater_or_equal"/>
<property name="result" channel="inline" abstraction="knowledge" model="domain.flag"/>
</part>
</model>
<model>
<part name="if-then-example" channel="inline" abstraction="operation" model="branch">
<property name="criterion" channel="inline" abstraction="knowledge" model="domain.flag"/>
<property name="true" channel="inline" abstraction="knowledge" model="domain.true_model"/>
<property name="false" channel="inline" abstraction="knowledge" model="domain.false_model"/>
</part>
</model>

```

Figure 5.2: Condition Control Structure and Elements in C and CYBOL

being some kind of comparison operation. In order to execute code upon activation of a flag, a conventional comparison control structure needs to be split up into two independent blocks (figure 5.2), with the flag being the linking element. The flag which was set by a comparison operation is used for branching the control flow.

The second example shows how a classical *if-then* statement would be written in CYBOL. The corresponding operation is called *branch* and it expects three properties: a *criterion* flag and two models, of which one is executed in case the flag is *true* and the other is executed otherwise.

```

<model>
<part name="if-then-example" channel="inline" abstraction="operation" model="branch">
<property name="criterion" channel="inline" abstraction="knowledge" model=".domain.flag"/>
<property name="true" channel="inline" abstraction="knowledge" model=".domain.true_model"/>
<property name="false" channel="inline" abstraction="knowledge" model=".domain.false_model"/>
</part>
</model>

```

## 5.3 Special Examples

*XML* is used for representing data of very different domains, and a whole plethora of *XML* dialects exists. Two of them are mentioned following. The main purpose of the next examples, however, is to show how *CYBOL* can replace these.

### 5.3.1 Synchronous Execution

*MusicXML* [11] is a markup language *designed to represent musical scores, specifically common western musical notation from the 17th century onwards*. In principle, *CYBOL* could be used for this purpose as well. Of course, there are many details (additional properties) which would still have to be worked out in order to be able to correctly represent complete musical scores. As most models, the *Musical Work* displayed in figure 5.3 can be considered a hierarchy consisting of *Parts* (played/ sung by instruments/ voices). *Parts* in turn consist of *Measures*, which consist of *Notes*, which finally have a *Pitch* and sometimes *Lyric*.

The image shows a musical score for Franz Schubert's 'Ave Maria'. It consists of three staves. The top staff is a vocal line in G major, starting with a treble clef and a key signature of one sharp (F#). The lyrics are: 'A - - - ve Ma-ri - - - al! Jung - - - frau', 'A - - - ve Ma-ri - - - al! Un - - - be -', and 'A - - - ve Ma-ri - - - al! Bei - - - ne-'. The middle and bottom staves are piano accompaniment, with the middle staff in treble clef and the bottom staff in bass clef. The piano part features a rhythmic pattern of eighth and sixteenth notes.

Figure 5.3: Musical Score of Franz Schubert's *Ave Maria* [11]

The following knowledge templates deliver only short examples showing how music may be modelled in *CYBOL*. Their property names were taken over from *MusicXML*'s element tags, as elaborated in [11]. Most are self-explanatory and shall not be further discussed here. The

first example template represents an extract from a complete musical *Work*, consisting of the two parts *Voice* and *Piano*:

```
<model>
  <part name="number" channel="inline" abstraction="string" model="D. 839"/>
  <part name="title" channel="inline" abstraction="string" model="Ave Maria (Ellen's Gesang III)"/>
  <part name="composer" channel="inline" abstraction="string" model="Franz Schubert"/>
  <part name="poet" channel="inline" abstraction="string" model="Walter Scott"/>
  <part name="voice" channel="file" abstraction="cybol" model="voice.cybol">
    <property name="score_instrument" channel="inline" abstraction="string" model="P1-I14"/>
    <property name="instrument_name" channel="inline" abstraction="string" model="Choir Aahs"/>
    <property name="midi_instrument" channel="inline" abstraction="string" model="P1-I14"/>
    <property name="midi-channel" channel="inline" abstraction="integer" model="1"/>
    <property name="midi-program" channel="inline" abstraction="integer" model="53"/>
  </part>
  <part name="piano" channel="file" abstraction="cybol" model="piano.cybol">
    <property ...
  </part>
</model>
```

One of the *Parts* is shown in the next template. It consists of several measures:

```
<model>
  <part name="measure_$1" channel="file" abstraction="cybol" model="measure_1.cybol">
    <property name="divisions" channel="inline" abstraction="integer" model="48"/>
    <property name="key_fifths" channel="inline" abstraction="integer" model="-2"/>
    <property name="key_mode" channel="inline" abstraction="string" model="major"/>
    <property name="beats" channel="inline" abstraction="integer" model="4"/>
    <property name="beat_type" channel="inline" abstraction="integer" model="4"/>
    <property name="staves" channel="inline" abstraction="integer" model="0"/>
    <property name="clef_sign" channel="inline" abstraction="string" model="G"/>
    <property name="clef_line" channel="inline" abstraction="integer" model="2"/>
  </part>
  <part name="measure_$2" channel="file" abstraction="cybol" model="measure_2.cybol">
    <property ...
  </part>
</model>
```

A *Measure* again consists of *Notes*:

```
<model>
  <part name="note_$1" channel="file" abstraction="cybol" model="note_1.cybol">
    <property name="duration" channel="inline" abstraction="integer" model="72"/>
    <property name="voice" channel="inline" abstraction="integer" model="1"/>
    <property name="type" channel="inline" abstraction="string" model="quarter"/>
  </part>
```

```

    <property name="stem" channel="inline" abstraction="string" model="down"/>
    <property name="position" channel="inline" abstraction="integer" model="1"/>
  </part>
  <part name="note_$2" channel="file" abstraction="cybol" model="note_2.cybol">
    <property name="duration" channel="inline" abstraction="integer" model="12"/>
    <property name="voice" channel="inline" abstraction="integer" model="1"/>
    <property name="type" channel="inline" abstraction="string" model="16th"/>
    <property name="stem" channel="inline" abstraction="string" model="up"/>
    <property name="position" channel="inline" abstraction="integer" model="2"/>
  </part>
  <part name="note_$3" channel="file" abstraction="cybol" model="note_3.cybol">
    <property ...
    <property name="position" channel="inline" abstraction="integer" model="2"/>
  </part>
</model>

```

An important property to note here is the *position* value. It is common that two notes have to be played at the same time, the notes then being called a *Chord*. In contrast to MusicXML which provides an own tag to denote notes belonging to the same chord, CYBOL suggests to use a *position* property having identical values for all notes in a chord. An interpreter program may thus not only read necessary sequence information, but can also figure out which of the notes have to be played *synchronously*.

A fourth example represents one *Note*, consisting of a *Pitch* and *Lyric* text, which are the final abstractions in this knowledge template:

```

<model>
  <part name="pitch" channel="inline" abstraction="string" model="B">
    <property name="alter" channel="inline" abstraction="integer" model="-1"/>
    <property name="octave" channel="inline" abstraction="integer" model="4"/>
  </part>
  <part name="lyric" channel="inline" abstraction="string" model="A">
    <property name="syllabic" channel="inline" abstraction="string" model="begin"/>
  </part>
</model>

```

### 5.3.2 Presentation and Content

The *Mathematical Markup Language* (MathML) [4] provides means for representing mathematical expressions, that is *Content* as well as *Presentation* of data. Both are discrete

models, comparable to the *Domain* and *User Interface* (UI) of a software application, which can be translated into each other.

CYBOL uses just four tags (section 2.3) but can represent mathematical expressions as well. What MathML calls *Content*, becomes a *Logic* knowledge template in CYBOL. The mathematical content of the formula  $(a + b)^2$  would be modelled as follows:

```
<model>
  <part name="addition" channel="inline" abstraction="operation" model="add">
    <property name="abstraction" channel="inline" abstraction="character" model="integer"/>
    <property name="summand_1" channel="inline" abstraction="knowledge" model=".domain.a"/>
    <property name="summand_2" channel="inline" abstraction="knowledge" model=".domain.b"/>
    <property name="sum" channel="inline" abstraction="knowledge" model=".domain.c"/>
  </part>
  <part name="exponentiation" channel="inline" abstraction="operation" model="power">
    <property name="base" channel="inline" abstraction="knowledge" model=".domain.c"/>
    <property name="power" channel="inline" abstraction="integer" model="2"/>
    <property name="result" channel="inline" abstraction="knowledge" model=".domain.r"/>
  </part>
</model>
```

And the formula's *Presentation* would be defined by the following two CYBOL *State* knowledge templates, of which the second one represents the *Base* that is referenced by the first one:

```
<model>
  <part name="base" channel="file" abstraction="compound" model="domain/base.cybol">
    <property name="fence" channel="inline" abstraction="boolean" model="true"/>
  </part>
  <part name="power" channel="inline" abstraction="integer" model="2">
    <property name="superscript" channel="inline" abstraction="boolean" model="true"/>
  </part>
</model>

<model>
  <part name="summand_$1" channel="inline" abstraction="character" model="a"/>
  <part name="operator" channel="inline" abstraction="character" model="+"/>
  <part name="summand_$2" channel="inline" abstraction="character" model="b"/>
</model>
```

### 5.3.3 Hello World

Two possible CYBOL versions of the famous minimal *Hello, World!* program are given following. The first consists of only two operations: *send* and *exit*. The string message to be displayed on screen is handed over as *property* to the *send* operation, before the *exit* operation shuts down the system:

```
<model>
  <part name="send_model_to_output" channel="inline" abstraction="operation" model="send">
    <property name="language" channel="inline" abstraction="character" model="shell"/>
    <property name="message" channel="inline" abstraction="character" model="Hello, World!"/>
  </part>
  <part name="exit_application" channel="inline" abstraction="operation" model="exit"/>
</model>
```

The second example template is slightly more complex. It starts with creating a domain model that consists of just one *greeting* string. That string is then sent as message to the human user via a *Textual User Interface* (TUI), just as in the first example. The difference is that now, the greeting is not handed over as hard-coded *string* value, but is read from the runtime knowledge model, which is indicated by its *abstraction* value:

```
<model>
  <part name="create_greeting" channel="inline" abstraction="operation" model="create">
    <property name="name" channel="inline" abstraction="character" model="greeting"/>
    <property name="abstraction" channel="inline" abstraction="character" model="character"/>
    <property name="element" channel="inline" abstraction="character" model="part"/>
    <property name="whole" channel="inline" abstraction="knowledge" model=".app"/>
  </part>
  <part name="receive_greeting" channel="inline" abstraction="operation" model="receive">
    <property name="channel" channel="inline" abstraction="character" model="inline"/>
    <property name="language" channel="inline" abstraction="character" model="character"/>
    <property name="message" channel="inline" abstraction="character" model="Hello, World!"/>
    <property name="model" channel="inline" abstraction="knowledge" model=".app.greeting"/>
  </part>
  <part name="send_model_to_output" channel="inline" abstraction="operation" model="send">
    <property name="language" channel="inline" abstraction="character" model="shell"/>
    <property name="message" channel="inline" abstraction="knowledge" model="greeting"/>
  </part>
  <part name="destroy_greeting" channel="inline" abstraction="operation" model="destroy_part">
    <property name="name" channel="inline" abstraction="knowledge" model="greeting"/>
  </part>
  <part name="exit_application" channel="inline" abstraction="operation" model="exit"/>
</model>
```



The appearance of a *create/destroy* pair in the second example already suggests how an application lifecycle with *startup*-, *runtime*- and *shutdown* phase could look like in CYBOL.

## 5.4 Inheritance as Property

One fundamental concept of *Object Oriented Programming* (OOP) is *Inheritance*. In principle, there is no problem with implementing inheritance in CYBOL. If done, however, it would differ from traditional class architectures as known from OOP. Classical OOP systems resolve inheritance relationships at runtime; CYBOP systems, on the other hand, would resolve them just once when creating a knowledge model (instance) from a knowledge template. After instantiation, all inheritance relationships are lost since instances are stored as purely hierarchical *whole-part* models in memory, without any links to *super* models.

The following knowledge template shows how inheritance could be realised in CYBOL. Contrary to OOP classes which hold a link to their corresponding *super* class as *intrinsic* property, a CYBOL knowledge template does not know itself from which *super* template to inherit from. That information is stored as *extrinsic* property outside the template instead, in other words in the *whole* template to which the inheriting template belongs.

```
<model>
  <part name="ok_button" channel="file" abstraction="cybol" model="gui/ok_button.cybol">
    <property name="super" channel="file" abstraction="cybol" model="button.cybol"/>
    <property name="size" channel="inline" abstraction="integer" model="90,30,1"/>
    <property name="colour" channel="inline" abstraction="rgb" model="127,127,127"/>
  </part>
</model>
```

One of the properties in the example template above carries the name *super*. Its model references another template which is treated as super template of the corresponding *part* the property belongs to. With slight modifications on the property name *super*, which has to be unique among all properties of a part, it would even be possible to implement *Multiple Inheritance*. Dependency complications are not to be expected because all inheritance relationships are forgotten in runtime models.

Although the described inheritance mechanism was tested successfully in an older prototype application, it has not been implemented in CYBOL. None of the created example

applications showed a need for it, nor did any of them promise more effective programming. The reuse of CYBOL templates is realised through composition only, that is fine-granular templates make up more coarse-grained ones. This counts for both, state- as well as logic models, since they are not bundled like in OOP. And polymorphism as effect does not have to be considered.

## 5.5 Container Mapping

State-of-the-art programming languages like Java offer a number of different container types (figure 5.4).

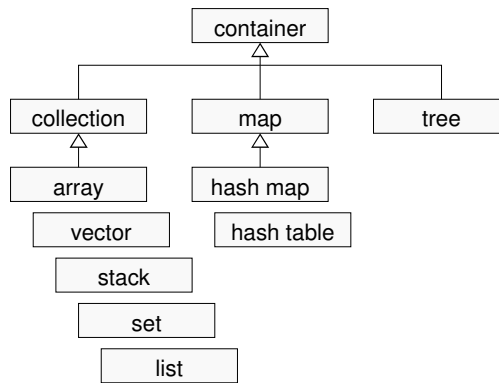


Figure 5.4: Classical Container Types in Java

CYBOI owns a *Knowledge Schema* which represents each item as *Hierarchy* by default, the result being that different types of containers are *not* needed any longer, that is are unified. Table 5.1 shows how the different kinds of container behaviour are implemented in CYBOL. As can be seen, CYBOL is able to represent many container types.

Classical Container Type	Realisation in CYBOL Knowledge Template
Tree	Hierarchical <i>whole-part</i> structure
Table	Like a Tree, as hierarchy consisting of rows which consist of columns
Map	Parts have a <i>name</i> (key) and a <i>model</i> (value)
List	Parts may have a <i>position</i> property
Vector	A <i>model</i> attribute may hold comma-separated values; an extra template holds a dynamically changeable number of parts
Array	Like a Vector; characters are interpreted as <i>string</i>

Table 5.1: Mapping Classical Containers to CYBOL

## 5.6 Hidden Patterns

There are a number of software patterns that may not be obvious (hidden) at first sight, but have been considered in the design of the CYBOL language.

Most obviously, CYBOL knowledge templates follow the *Composite* pattern, in a simplified form. All templates represent a compound consisting of part templates, which leads to a tree-like structure. But this also means that related patterns like *Whole-Part* and *Wrapper* are representable by CYBOL knowledge templates. A template as whole wraps its parts.

Knowledge templates with similar granularity can be collected in one directory, in other words one common ontological level. Templates with smaller granularity, that is those that the more coarse-grained templates consist of, can be placed in another common layer and so forth. What comes out of it is a system of levels – one variant of the *Layers* pattern.



## 6 Diagrams

Because of the different programming philosophy behind CYBOP, standard *Unified Modeling Language* (UML) diagrams cannot be used unalteredly for the design of CYBOL applications. Some of them, however, could be quite useful, when adapted a bit. For creating CYBOL applications, the following four can be considered sufficient. They model the structure of:

1. *Template Diagram* (TD): one design-time template (hierarchical, ontological concept), with purely unidirectional relations; does not illustrate relations between different concepts, as these are only established by logic models at runtime; could look like a UML class diagram (CsD) or a tree, only that a template may not only represent states, but also logic (algorithms, workflows) (figure 6.1)
2. *Model Diagram* (MD): the runtime model tree; comparable to UML object diagram, but a simple tree with named nodes would suffice; is important because input/ output parameters of operations are given as dot-separated paths to runtime knowledge tree models (figure 6.2)
3. *Organisation Diagram* (OD): template directories; could look like a UML component- or package diagram or a simple tree (figure 6.3)
4. *Communication Diagram* (CD): a network of communicating systems, which may run on the same or on different physical machines (nodes); could look like a UML distribution diagram; not to be mixed up with UML collaboration diagram (figure 6.4)

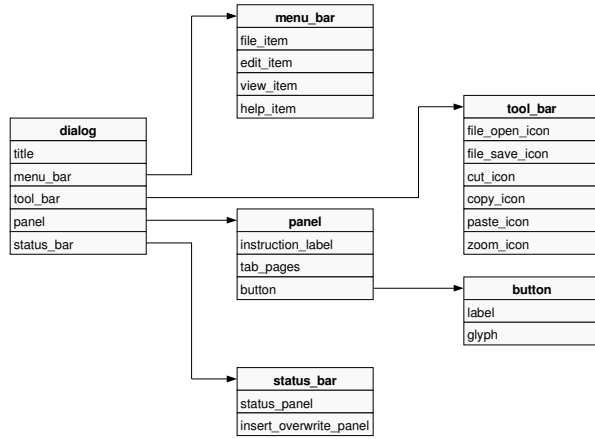


Figure 6.1: CYBOL Template Diagram (TD) Proposal

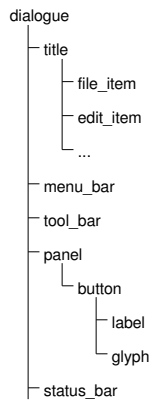


Figure 6.2: CYBOL Model Diagram (MD) Proposal

As said above, the four diagrams may look similar to their corresponding UML pendant. One possible proposal is given for each diagram type. The TD in figure 6.1 illustrates a graphical dialogue. The diagram looks pretty similar to a UML CsD. Attributes and methods are not bundled in one concept though, and inheritance does not exist. Associations are drawn if a concept links to an external concept which may reside in another file (like the *menu\_bar*), for example. If a part (like the *title*) is hold inline in the concept, on the other hand, an association is not displayed. Upon clicking on a part in a concept box, a dialogue opens up that allows the entry of meta data like the part's channel, abstraction, model and further properties (details).

The MD in figure 6.2 displays the runtime models that were instantiated with knowledge templates providing the initial values. Again, the parts of a graphical dialogue were used.

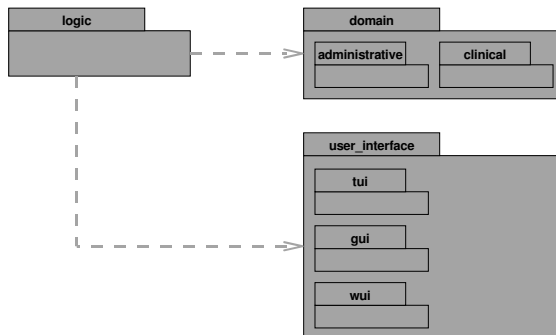


Figure 6.3: CYBOL Organisation Diagram (OD) Proposal

The OD in figure 6.3 shows packages into which CYBOL knowledge templates may be organised. Packages do normally correspond to directories on file system level. The figure contains a *domain* package consisting of two sub packages, one containing knowledge templates for *administrative* patient data and the other holding templates for *clinical* data of a patient. Also, there is a *User Interface* (UI) package containing three sub packages, for: *Textual UI* (TUI), *Graphical UI* (GUI) and *Web UI* (WUI). Both, *domain*- as well as *user\_interface* packages may be accessed from the operations residing in the *logic* package.

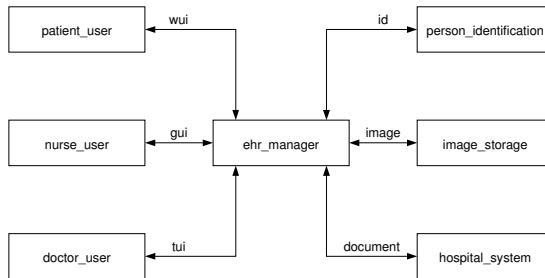


Figure 6.4: CYBOL Communication Diagram (CD) Proposal

The CD in figure 6.4, finally, shows a number of independent systems communicating with each other. An *Electronic Health Record* (EHR) manager application may be found in the center of the figure. Patients communicate with it using a WUI; nurses using a GUI and doctors using a TUI (for better performance). A patient gets identified by asking a *person\_identification* service. Documents may be exchanged with a *hospital\_system* and images with a special *image\_storage* system.



## 7 Appendices

### 7.1 Abbreviations

BNF	Backus Naur Form
CD	Communication Diagram
CsD	Class Diagram
CYBOI	Cybernetics Oriented Interpreter
CYBOL	Cybernetics Oriented Language
CYBOP	Cybernetics Oriented Programming
DB	Database
DBMS	DB Management System
DTD	Document Type Definition
EBNF	Extended BNF
EHR	Electronic Health Record
EPS	Encapsulated PS
FDL	Free Documentation License
GNU	GNU is not UNIX
GPL	General Public License
GUI	Graphical UI
HTML	Hypertext ML
i/o	input/ output
MathML	Mathematical ML
MD	Model Diagram
ML	Markup Language
OD	Organisation Diagram
OO	Object Oriented

---

	(Object Orientation)
OOP	OO Programming
OS	Operating System
PS	PostScript
RAM	Random Access Memory
SPP	Structured and Procedural Programming
TD	Template Diagram
TUI	Textual UI
UI	User Interface
UML	Unified Modeling Language
UNIX	Universal Interactive Executive (Uniplexed Information and Computing System)
VM	Virtual Machine
WUI	Web UI
XML	Extensible ML
XSD	XML Schema Definition

## 7.2 References

- [1] John Backus, Peter Naur, and et al. Revised report on the algorithmic language algol 60. In Peter Naur, editor, *Communications of the ACM*, volume 3, no. 5, pages 299–314, May 1960. <http://www.masswerk.at/algol60/report.htm>.
- [2] Wissenschaftlicher Rat der Dudenredaktion: Guenther Drosdowski ..., editor. *Der Duden: in 12 Baenden; das Standardwerk zur deutschen Sprache*, volume Duden, Rechtschreibung der deutschen Sprache. Dudenverlag, Mannheim; Leipzig; Wien; Zuerich, 21st edition, 1996. <http://www.duden.de/>.
- [3] James Gosling, Bill Joy, Guy Steele, and et al. *The Java Programming Language Specification; The Java Development Kit (JDK)*. Sun Microsystems, Inc., Santa Clara, 2nd edition, 1996-2000. <http://java.sun.com>.
- [4] World Wide Web Consortium (W3C) Math Working Group. Mathematical markup language (mathml) 2.0 recommendation. Online Specification, October 2003. <http://www.w3.org/TR/2003/REC-MathML2-20031021/>.
- [5] Christian Heller. Cybernetics oriented programming (cybop) in res medicinae. In *OSHCA Conference Online Proceedings*, Los Angeles, November 2002. Open Source Health Care Alliance (OSHCA). <http://www.oshca.org/>.
- [6] Christian Heller. Cybernetics oriented language (cybol). *IIIS Proceedings: 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004)*, V:178–185, July 2004. <http://www.iiisci.org/sci2004> or <http://www.cybop.net>.
- [7] Christian Heller. *Cybernetics Oriented Programming (CYBOP) – An Investigation on the Applicability of Inter-Disciplinary Concepts to Software System Development*. Tux Tax, Ilmenau, January 2007. <http://www.tuxtax.de>.
- [8] Denis Howe. Free on-line dictionary of computing (foldoc). Internet Database, September 2003. <http://wombat.doc.ic.ac.uk/foldoc/Dictionary.gz>, <http://www.foldoc.org/>.
- [9] Markus Guenther Kuhn. A summary of the iso ebnf notation. Web Document, September 1998. <http://www.cl.cam.ac.uk/mgk25/iso-ebnf.html>.
- [10] The linux documentation project. HOWTOs, Guides, FAQs, man pages, Linux Gazette, LinuxFocus, 2004. <http://www.tldp.org/>.
- [11] Recordare LLC. Musicxml definition 1.0. Online Specification, March 2005. <http://www.musicxml.org>.

- [12] The Mentor. The hacker manifesto, January 1986. <http://www.phrack.org/archives/7/P07-03>.
- [13] CYBOP Project. Cybernetics oriented programming (cybop), 2002-2004. <http://www.cybop.net>.
- [14] Debian Project. Debian gnu/linux, 1997-2004. <http://www.debian.org>.
- [15] Refsnes Data. *W3Schools - Full Web Building Tutorials - All Free*, 1999-2004. <http://www.w3schools.com/>.
- [16] Norman Walsh and Leonard Mueller. *DocBook: The Definitive Guide*. O'REILLY, <http://www.oreilly.com/>, v4.3cr3 edition, January 2004. <http://docbook.org/>.
- [17] World Wide Web Consortium (W3C). *Extensible Markup Language (XML)*, 1.0 edition, February 1998. <http://www.w3.org/TR/1998/REC-xml-19980210>.

## 7.3 Figures

1.1	CYBOL Interpretation . . . . .	2
2.1	Recommended CYBOL DTD . . . . .	5
2.2	Simplified CYBOL DTD . . . . .	5
2.3	Simplified CYBOL XSD . . . . .	6
2.4	Recommended CYBOL XSD . . . . .	7
2.5	CYBOL in EBNF . . . . .	8
3.1	Textual User Interface . . . . .	14
3.2	Graphical User Interface . . . . .	20
3.3	Web User Interface . . . . .	25
5.1	Loop Control Structure and Elements in C and CYBOL . . . . .	78
5.2	Condition Control Structure and Elements in C and CYBOL . . . . .	79
5.3	Musical Score of Franz Schubert's <i>Ave Maria</i> [11] . . . . .	80
5.4	Classical Container Types in Java . . . . .	86
6.1	CYBOL Template Diagram (TD) Proposal . . . . .	90
6.2	CYBOL Model Diagram (MD) Proposal . . . . .	90
6.3	CYBOL Organisation Diagram (OD) Proposal . . . . .	91
6.4	CYBOL Communication Diagram (CD) Proposal . . . . .	92



---

## 7.4 Tables

1.1	Analogy between the Java- and CYBOP World . . . . .	2
5.1	Mapping Classical Containers to CYBOL . . . . .	87





## 7.5 History

### TODO

- Rename and reduce standard tags: model, part, property, constraint into just: compound, part, meta
- Rename two of the four standard attributes: name, channel, abstraction, model into: name, channel, language, message

### 2.0 (2007-07-31)

- Release CYBOL definition as independent specification document
- Split "receive" operation: a model has to be created with "create" first before it can be handed over to "receive", to be filled with data
- Change many properties of state- and logic models

### 1.0 (2005-12-12)

- Release initial CYBOL definition within the CYBOP book [7]

### 0.x (2002 to 2004)

- Publicise paper on CYBOL [6]
- Experiment with several XML representation formats
- Introduce first CYBOP ideas to the public [5]



## 7.6 Licences

### 7.6.1 GNU General Public License

#### Version 2, June 1991

Copyright © 1989, 1991. Free Software Foundation, Inc.

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

#### PREAMBLE

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## **TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying,

distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

#### **NO WARRANTY**

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH



ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

### **How to Apply These Terms to Your New Programs**

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

One line to give the program's name and an idea of what it does. Copyright (C) yyyy name of author

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other

than ‘show w’ and ‘show c’; they could even be mouse- clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

## 7.6.2 GNU Free Documentation License

### Version 1.2, November 2002

Copyright © 2000, 2001, 2002. Free Software Foundation, Inc.

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## TERMS AND CONDITIONS

### PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

### APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept

the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format,  $\text{\LaTeX}$  input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modifica-

tion. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section *Copying in Quantity*.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

## MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections *Verbatim Copying* and *Copying In Quantity* above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## **COMBINING DOCUMENTS**

You may combine the Document with other documents released under this License, under the terms defined in section *Modifications* above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents,



unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

## **COLLECTIONS OF DOCUMENTS**

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## **AGGREGATION WITH INDEPENDENT WORKS**

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section *Copying In Quantity* is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the

Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section *Modifications*. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section *Modifications*) to Preserve its Title (section *Applicability and Definitions*) will typically require changing the actual title.

## TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

## FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

### **How to use this License for your documents**

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.



## 7.7 Index

- abstraction Attribute, 4, 9
- Add, 47
- Algorithm Division Example, 76
- Analogy between CYBOP and Java, 2
- AND, 37
- Archive File, 65
- Arithmetic, 47
- Attributes, 9
  
- Backus Naur Form, 6
- Bit Manipulation, 29
- BNF, 6
- Boolean Logic, 35
- Branch, 40
  
- CD, 89
- channel Attribute, 4, 9
- Communication, 59
- Communication Diagram, 89
- Compare, 45
- Comparison, 45
- Conditional Execution Example, 78
- constraint Tag, 4, 10
- Containers in CYBOL, 86
- Copy, 54
- Copy File, 67
- Count, 42
- Create, 52
- CYBOI as Virtual Machine, 2
- CYBOL Logic Example Constructs, 75
- CYBOL Special Example Constructs, 80
- CYBOP, 1
- CYBOP-Java Analogy, 2
  
- Debian GNU/Linux Package Definition, 74
- Definition, 3
- Destroy, 54
- Diagrams, 89
- Divide, 51
- DocBook DTD, 71
- Document Type Definition, 4
- DTD, 4
  
- EBNF, 4, 6
- Encapsulated PostScript, 73
- EPS, 73
- Examples, 71
- Execute File, 68
- Exit, 58
- Extended Backus Naur Form, 4, 6
- Extensible Markup Language, 3, 4, 6, 9
- External Resources Example, 73
  
- Get, 44
- Get Bit, 34
- Grammar, 3
- Graphical User Interface, 19, 72
- GUI, 72
- GUI Layouts, 72
  
- Hello, World
  - Example, 84
- Hidden Patterns in CYBOL, 87
- History, 101
- HTTP, 73
- Hyper Text Transfer Protocol, 73
  
- Inheritance as CYBOL Property, 85
- Interrupt, 65
- Introduction, 1

- Java Swing Framework, 72
- Java-CYBOP Analogy, 2
- Lifecycle Management, 56
- List File, 68
- Logic Knowledge Modelling, 9
- Logic Models, 29
- Loop, 42
- Loop as Operation Example, 77
- Mapping Containers to CYBOL, 86
- Markup Tag, 4
- Mathematical Markup Language, 82
- MathML, 82
- MD, 89
- Memory Management, 52
- Meta Constraints Example, 74
- Meta Data Hierarchy, 3
- Meta Property Example, 72
- model Attribute, 4, 9
- Model Diagram, 89
- model Tag, 4, 10
- Model-Part Relation Example, 71
- Move, 55
- Multiply, 50
- MusicXML, 80
- name Attribute, 4, 9
- NEG, 36
- NOT, 35
- OD, 89
- Operation Call Example, 75
- OR, 38
- Organisation Diagram, 89
- part Tag, 4, 10
- Patterns in CYBOL, 87
- Presentation and Content Example, 82
- Program Flow, 40
- property Tag, 4, 10
- Receive, 62
- Reset Bit, 33
- Rotate, 30
- Semantics, 9
- Send, 59
- Serialised Model Example, 73
- Set Bit, 32
- Shell Commands, 65
- Shift, 29
- Shutdown, 58
- Simple Assignment Example, 76
- Startup, 56
- State Examples, 71
- State Knowledge Modelling, 9
- State Models, 13
- Subtract, 48
- Symbols, 4
- Synchronous Execution Example, 80
- Syntax, 3
- Tag-Attribute Swapping, 11
- Tags, 10
- TD, 89
- Template Diagram, 89
- Terms, 4
- Textual User Interface, 13
- The Linux Documentation Project, 71
- Theory, 1
- TLDP, 71
- UML, 89

Unified Modeling Language, 89

User Interface, 13

Vocabulary, 4

Web User Interface, 24

Whole-Part Hierarchy, 3

XML, 3, 4, 6, 9

XML Attribute, 3, 9

XML Schema, 6

XML Schema Definition, 4, 6

XML Tag, 3, 9

XOR, 39

XSD, 4, 6