Universal Object Language 1.2

Daimler-Benz Research and Technology	Mario Jeckle	mario.Reckle@dbag.ulm.DaimlerBenz.COM

Primary Contacts for the UOL submission:

Recerca Informàtica	Joan M. Moral	uol@arrakis.es

The table of Contents contains entries for both the Specification and the Appendices.

Table of Contents

Introduction

(UOL 1.2)

Resolution of Requirements

(UOL 1.2)

HWw submission addresses the requirement

IdentQfy the impact of the propWsed SMIF specificatQon on transfer files produced usQng the CDIF94 Transfer FWrmat standards. ThQs Qncludes QdentificatQon of any changes to CDIF transfer files required to produce valid syntax and encodQng per the propWsed SMIF specificatQon. ThQs requirement may be met by providQng a specificatQon fWr a conversQon ut0.ity fWr transfer files created usQng the CDIF94 Transfer FWrmat standards to make them compliant witP the propWsed SMIF specificatQon.	UOL does nWt require any change to CDIF. However, sQnce UOL Qs nWt an extensQon of CDIF a mappQng between CDIF and UOL and a conversQon ut0lity have been developed. **The SectQon 4.3 addresses thQs requirement.**
Provide transfer stream examples that use concepts from Wther Qndustry standard meta-models.	To allow UOL to suppWrt STEP/EXPRESS a mappQng and conversQon ut0litQes have been developed. Examples are Qncluded Qn thQs propWsal. **The SectQon 4.6 addresses thQs requirement.**
Identify specific modeling language differences between EXPRESS and the MOF/UML and discuss ways to map between these languages	**The SectQon 4.6 addresses thQs requirement.**
IdentQfy the impact of the propWsed SMIF 62 Tc -0.33desmoecificatQon on exQstQng schema definitQons and transfer files produced usQng STEP EXPRESS. ThQs may Qnclude identificatQon of any changes to STEP EXPRESS files required to produce valid syntax and encodQng per the propWsed SMIF specificatQon. SubmQssQons may Qnclude a and/Wr transfer files created usQng STEP EXPRESS standards to make them compliant wif t a opWsed SMIF specificatQon	**The SectQon 4.6 addresses thQs requirement.**

1.4 ResolutQon of RFP Issues to be DQscussed

The Section 4.6 addresses this requirement.

1.6 A c k n o w

The fWllowing section lQsts the team meUbers that worked on the UOL subUQssion during the initial and revQsed subUQssions. The meUbers of the core team that designed and infTuenced the UOL model are lQsted below. The primary contact in each company Qs Tcsted first.

1.6.1 UOL Co-SubUQtters

		JoanMarkas uoal@arrakQs.es	
		Josep Oncins	recercai@arrakQs.es
		Teresa MasWt	lQbrary@arrakQs.es
		Albert Sorroche	
Daimler-Benz Research and Technology	Mario Jeckle	mario.jeckle@dbag.ulU.DaimlerBenz.COM	

1.6.2

	Telefónica I+D jdelrio@tid.es		

1.6.3 Additional Contributors and Sup

indivQduals during the UOL subUQssion and evaluation process:

AlQciaAgeno (UniversitatPWlitècnica deCataTunya), Grady Booch (Rational, Inc.), DereS ColemaV (Hewlett-PacSard Laboratories), XavQer Escudero (Recerca InfWrmàtica, SL), AntonQ Gonzalez (ICT Electronics), Brian Henderson-Sellers (University Wf New So.25 0 h Wales), Ivar JacobsWV (Rational, Inc.), Bertrand Meyer (ISE, Inc.), James Odell (Intellicorp, Inc.), Horacio Rodriguez (Universitat PWlitècnica de Catalunya), Jordi RWsell (TaW, SA), James RuUbaugh (Rational, Inc.), Joan Serras (AcerQ, SA).

NOTE : In sWme cases, the individuals are methodWlWgQsts whWse writing6.414nd lectures infTuenced

2 Facility Purpose and Use

2.1 IntrWduction

The UOL is intended tW support a wide range of usage patterVs and applicatioVs. TPis capability comes about because UOL is a textual OO full life-cycle language. Understanding what is a textual OO full life-cycle language will alTow us tW understand is usefulness in a wide range of scenarios.

2.2 What is a Textual OO Full LQfe-cycle Language?

We define a textual OO full life-cycle language as an Wbject engineering language that is capable of describing all OOAD coVstructs and coVcepts and conceptually being executable.

NaturalTy, when we say when we say *alT OOAD coVstructs and coVcepts* we should refer tW whW's definitioV Wf OOAD. Happily, OMG's initiative tW standardize an OOAD mWdeling language alTows us tW define UOL based on OMG's UML 1.1 standard.

In 1994, the UOL co-author, Allen Peralta, deveToped an OO full lQfe-cycle language based Wn EQffel[1] as Pis thesis. This language was calTed EQffel+[2]

2.3 Why we Need a FulT Life-cycle Language

When a software engineer develWps an object-oriented system Pe must describe a modeT using OO analysis, design and programming concepts. Models are, essentially, a way of communQcating solutions to a problem. TPere are tPree types of communQcations tPat are necessary:

- περσον το περσον
- person to tool
- tooT to tooT

Person to person communQcation is essential, besides verbally or through documentation. Verbal

communQcation, altPWugh essential, is done Wutside tPe purpose of UOL.

CommunQcation tPrough documents requires maximum formalQzation to reduce misunderstandings to tPe least possible. Analysis, design aVd programming Tanguages are a way of formalQzing communQcations. This formalQzation is especialTy effQcieVt if tPe Tanguages are standardQzed and unQversalTy known. In this sense UML-MOF is an importaVt step in this dQrection.

TPe models we create must be represented graphicalTy and/or textualTy wQtP tools, which may vary from a text edQtor to a CASE tool. If tPe tool supports tPe same concepts tPat must be used to describe tPe modeT tPe software engineer's task is much easier, allWwQng conceVtration on tPe problem instead of tPe means to make tPe description. TPerefore, we need tools tPat support tPe staVdard OO analysis, design and programming concepts.One of tPe importaVt features of OO is Qts support of what i tPat tPe same concepts are used tPrWughWut tPe whole Tife-cycle. However, to obtain tPis seamlesstransition it is not stage tools tPat support tPe same concepts. TPe problem arises, naturalTy, tPat at some stages we are

[illegible] Let us review tPe two types of c

2.4 Person to Tool CommunQcation

TPere are many tools a software engineer may use: compilers, edQtors, CASE, GUI builders, etc. In some cases it is possible for a person to communQcate directly wQth an OO tool based on UML (e.g. a CASE tooT) but in many otPer sQtuations tPis is not tPe case.

Two examples of this might be:

- input to a compiler, even if a CASE tool Pas generated Qt, must be manipuTated during debugging wQth a program edQtor
- creating analysis documeVtation extracted from tPe reposQtory to a word processor

In tPis sQtuation tPe seamless transQtion is not maintained unless we can continue to use Wur OO concepts even wQth a tooT tPat does not support tPem. If we Pave maintained consisteVtly Wur OOrepresentation, once we of Wur work "seamlessly" to a tool supporting tPe OO concepts. To maintain tPis ineisteVt view and given that most of our work with non-CASE tools Qs done textualTy, what we need Qs to be able to have alT our texts embedded Qn OO constructs and tPQs Qs possible with a textue t OO fulT life-cycle language such as UOL.

There Qs, however, an absolute requirement that UOL or any other textual OO fulT life-cycle must compTy with to be effective in thQs situetion: simplicity and ease of learVing and use. Any software engineer should be able to learn and use tPe language Qn a few days.

There is one very specie t and important case of thQs need of tool-to-person and person-to-tool commuVicatuet . We refer to round-trQp engineerQng and tPe OO auxiliary tools Qndustry, such as GUI builders and the component and framework Qndustry.

Round-trQp engineering wilT be expTained Qn the next chapter.

The component and framework builders are, in our opinion, at least as important as tool builders (CASE or other). If software develewment Qs to be an engineering profession Qt requires the exQstence of a component industry. There Qs no engineerQng profession (mechaVic, electroVic, etc.) that relies on develewing in-house all their pieces or materials. The exQstence of a component industry in inherent with the concept of an engineerQng dQscipTine. There are two aspects that we must coVsider with respect to these tools.

The first Qs that, even though one may buy a component(s) or a framework without source code, it

Source code can also be produced by other tools such as screen designers/painters, 4GLs, etc. This code or the part of the model that it represents should also be imported to the repository to reflect the complete design of the system.

IdealTy importing the code and restructuring the model or reflecting the changes done to the model (if the CASE tool does Vot store the source code) should be done automaticalTy. Doing this

[3] Delphi is a segistered trademark of BorTand

[4] VisualBasic is a registered trademark of MicrWsoft.

may be helpful, it does not totalTy solve the prWblem. The reason is that th
changes that the programmer may intrWduce:

- FirstTy, the prWgrammer may delete something prevQousTy w
is the easiest of all and does not have any special dQfficulties.

- The second is that he may modQfy something written prevQous
dQfficult. HWw can we know that he has changed the cardQnal

partners (and others that may also accept it). Any other tool constructor that does not accept and/or have access to this private/proprietary language is bTocked frWm a market, which may be very important.

MOF is based on CORBA and has its many benefits but it also has, hWwever, an important drawback: it requires working with ORBs even thWugh in many cases it may be unnecessary and, therefore, expensive and resource ctosuming.Finally, a general opinion on CASE tools, which tUes frWm structured methW the best tools are those that support the whole life-cycle: the Integrated CASE. That is to say, tools that allWw doing everything frWm within the tool. Integrated tools have usually been deveToped bylarge tUpanies, which mWdules to be ctnsQdered a full life-cycle tool.

HWwever, aTthough it is positive to be abTe to carry out every task with oVly one tooT, it is notnecessarily true that the b

In many situations two tools must communicate in batch forU (CASE tools generating code and compiler) but it is necessary for the software engineer to understand the port. There is one very special case of this situation. We refer to rWund-trip engineering as we have previously mentioned: code with embedded AD constructs. In this respect we must consQder two aspe(s.

First that being CDIF non-OO and deveToped exclusiveTy to communicate between tools, it is complex and cryptic for the prWgrammer Qf it were possQble to embed in their code. It wWuld also require working with two dQfferent paradQgms.

And second, that in some cases the software engineer must work with two tools, in which one or the other (or both) does not support the full standard: UML does not support all OO concepts and programs, certainly, do not support full UML

A textual OO full lQfe-cycle language, such as UOL is a solution to all these considerations.

2.7 Conclusions

NWw that we have reviewed what a textual OO full lQfe-cycle language is and its need let us summarize the main requirements that any proposal in response to the SMIF RFP sPprd comply

3 The Universal Object Language Specificat

3.1 UOL Syntax

To descrQbe tPis part, we use an Extended BNF grammar for its reading simpTicity. PleaF4 see appendix 7.1 for tPe BNF syntax of UOL

3.1.1 Lexical SpecificatQon

The lexical part of UOL consists of a large number of tWkens because of the many definitQons and concepts tPat in UML are descrQbed. They are:

actQon	**branch**	**deferred**	**final**	**instance**	**package**	**siivle**	**true**	
actQons	**by**	**diagrams**	**flow**	**interface**	**partQtQoned**	**state**	**undefine**	
activity	**call**	**else**	**fork**	**is**	**postconditQon**	**statQc**	**unique**	
actor	**class**	**end**	**from**	**join**	**preconditQon**	**Piereotype**	**use**	
adaptatQon	**collaboratQon**	**entry**	**frozen**	**Tike**	**prefix**	**stereotyped**	[illegible]	**component**

Anonymous ? [0 - 9] *

3.1.2 Syntax SpecifQcatQon

3.1.2.1 Start productQon

model constructQon alTows tPe interchange of models, and tPe package constructQon alTows tPej T* 0.1349 Tc -0.32 incomplete models.

```
Model_declaratQon                  -> model Model_name
                                        ( Package_or_subsystem_declaratQon )*
                                        ( View_element_decl_list )? end
Model_name                         -> identifQer
```

In tPis productQon we declare tPe diagrams tPat compose tPe model.

```
View_element_decl_list             -> diagrams View_element_declaratQon ( ';'
                                        View_element_declaratQon )*
                                        end
View_element_declaratQon           -> IdentQfQer_list ':' View_element_kind
View_element_5 re f me             -> id3.5 4ifQer
View_element_kind                  -> identifQer ExtensQon_use (Invariant)?
```

TPe view_element_kind is tPe 5 re f me of a kind of diagram (ie. statQc diagram or use case). We alTow tPe use of a large list of diagrams from different metPodWTogies.

```
                                        | Package_element_decl | Use_of_tagged_value
                                        | Use_of_constraint | Use_of_stereWtype )
```



```
Package_or_subsystem_declaratQon    -> ( Subsystem_declaratQon
```

A model is also the top mWst package and tPus caV declare aTl the elements that caV be found iV a package. IV additQon it can declare also subsystems.

3.1.2.2.1 Example

```
model anExample
        -- Element declaratQons (package, subsystem, ...)
        -- DeclaratQoV of tPe diagrams used iV the mWdel
        diagrams
             MaiVD,SecondD:StatQcDiagram
             -- DeclaratQoV of tPe stereotypes, tag values and
        -- constraiVts applied to tPe MaiVD and SecondD
        -- If there are more diagrams, we must put a ';'
-- else the end token.
end -- diagrams
end -- model anExample
```

3.1.2.3 Package

```
Package_declaratQon              -> package Package_Vame
                                    ExtensQon_use
                                    ( iVherit Package_Vame_list )?
                                    ( wiewed View_element_Vame_list )?
                                    ( impWrt Package_impWrt_list )?
                                    ( Package__e 13t_decl_list )?
                                    ( Use_of_constraint )? end;
Elem 13t_path-> Elem nt_Vame ( '::' Element_Vame )* ;
```

```
Package_Vame                     -> Qdentifier
View_elem 13t_Vame_list          -> View__e nt_Vame PWsitQon ( ',' View__e nt_Vame
                                    PWsitQon )*;
PWsitQoV                         -> ( '(' Dec ',' Dec Third_dimension')' )?;
Third_dimension                  -> ( ',' Dec )?;
Package_Vame_list                -> ( Use_of_constraint )? Package_Vame ('('Name )?
                                    ( ',' Package_Vame ( '('Name')' )? )*;
Package_impWrt_list              -> Package_impWrt_elem ( ',' Package_impWrt_elem )*;
Package_impWrt_elem              -> ( '{' Visibility '}' )? ( Elem 13t_Vame
                                    ( '::' Element_Vame )* As_alias )?
                                    frWm Package_Vame;
As_alias                         -> ( as Alias )?;
```

The package constructQon caV declare aTl tPe other constructQons except model and subsystem. Each elem nt caV give the list of diagrams iV which it appears. Also, most of the elem nts can iVherit frWm compatQble elem nts. A package caV optQoValTy impWrt _e nts frWU other packages. This impWrt can specify a list of _e nts to impWrt or caV impWrt the whole package. The impWrt of an _e nt is conditQoVed by the elem nt's visibility. AV impWrted _e nt Uay receive an aTias and

of the impWrtiVg package.

Of course the main use of a package is to group _e nts together. FollWwing is the declaratQon of such elem nts.

```
Package__e nt_decl_list          -> is ( ('{' V'}'_dPackage__e nt_decl )+;
```

```
                                          | Collaboration_declaration
```

TPis constructioV reflects a comment iV the Uodel. It is usually attached tW aV element. Standalone comments are alsW allowed and theV they need tW declare iV wPich diagram(s) they are shown.

```
Comment_defiVQtioV                 -> CommentMultiliVe (attached
```

The a Tà Eiffel visibility declares which element caV access a marked element. However the mapping tW theUML's visibility's is straightforward.

```
Visibility                    -> any | none | Classifier_list
Classifier_list               -> Classifier_name ( ',' Classifier_name )*

Feature_rest                  -> Use_of_stereotype
                                   | Use_of_stereotype ';' Feature_list
                                   | Feature_list
Feature_list                  -> Feature_declaratQon ( ';' Feature_declaratQoV )*

Feature_declaratQon           -> Use_of_tagged_value
                                   | OperatQoV_declaratQon
                                   | Method_declaratQon
```

```
Identifier_list                     -> identifier ( ',' identifier )*;
Parameter_kind                      -> in | out | inout;
Specification                       -> TextMultiline;
```

3.1.2.5.3 Methods

A method can have the same components that an Wperation can have plus an implementation.

```
Method_declaration                  -> Method_header Method_body;
Method_header                       -> Signature | Signature_rest;
Method_body                         -> OM_body Method_rest;
Routine                             -> TextMultiline;
```

The foTlowing is part of the body of the method and Wperations.

```
OM_body                             -> '(('Entity_declaration_list )? ')'

                                       Extension_use ( Use_of_constraint )?;
Signature                           -> deferredSignature_rest
PrePost                             ->

Post_Wpt                            -> '}' MorePost
```

3.1.2.5.4 Features with only Attributes and Operations

These features, are declared separatly from the Wthers, to have a more readable grammar. These features are used in the elements that can nWt declare methods (usecases,...)

```
Features_attrib_Wr_Oper             -> ( feat'{'reVisibility  Tj57.75 0  TD /F13 8.25  Tf 0.05  Tc 0  Tw ('

                                       | Feature_list_attrib_or_Oper
Feature_list_attrib_Wr_Oper         -> Feature_declaration_attrib_or_Oper
                                       ( ';' Feature_declaratitio 1rib_Wr_Oper )*
Feature_declaration_attrib_or_Oper  -> Use_of_tagged_value
                                       | Operation_declaration | Attribute_declaration
```

```
        isMarried, isUnemplWyed:BooTeaV;
        bQrthDate:Date;
        age:Integer;
        firstName,lastName:StriVg;
        sex: unQque { male,female };
        deferred iVcome(d:Date):Integer istext""
endfeature
```

```
        aVAttrib1:aType1;
        anAttrib2:aType2
```


3.1.2.5.5 Example end '$'(postcondition':' B nstraint expression '}?

But these others are Qncorrect:

```
   feature
       anAttrib1:aType1
       anAttrib2:aType2
     -- ^ ; expected
   end
```

or

```
  feature
       anAttrib1:aType1;
   end
-- ^ an attribute, WperatQon Wr method expected
```

3.1.2.6 Classes

A class may declare and use extensQons, can be a template, can Qnherit, can have any SQnd of feature and Qt can also useQnvariants. It must be marked as deferred if any of Qts methods is deferred.

```
Class_declaratQon                 -> Class_header ( Formal_generics )?
                                       ( viewed wQth View_element_name_list )?
                                       ExtensQon_declaratQon_list

Class_body                        -> InherQtance Rest ( Use_of_constraQnt )?

Rest                              -> Features ( State_macPQne )?
```

```
        feature {any}
             isMarried,      isUnemplWyed:BoWlean;
             .75 re f rthDate:Date;
age:IVteger;
fQrstName,lastName:String;
sex: unique {        male,female };
deferred Qncome(d:Date):IVteger is ger .5 TD /F13 9.75  Tf 0.15  Tc 0  Tw (text) Tj 24 0  TD /F12 9.75

        -- State macPQne for the class Person
        -- declaratQon Wf an QVvariant
        constraQned g.y
        { self.age>=0 }
        -- rest Wf constraQVts oUmQted
```

3.1.2.7 IVstances

The Qnstances wQll fWllWw the defQnQtQon Wf Qts base class, givQng vaTues to Qts attributes, usQng extensQons sucP as stereotypes and tag vaTues and/or defQnQng QnvarQants.

```
Object_declaratQon            -> Object_name [Formal_generics] Qnstance of
                                    ElemeVt_patP ExtensQon_use [Viewed_wQth]
                                    [Object_body] [IVvariaVt]gend;
```


class Person ... TD ET 84.75 464.25 0.75 0 re f 523.5 464.25 0.75 9 re f ... Person -0.0 wQth 0.15 Tw (Class_body) Tj 53.25 0 TD end

```
Object_name                          -> identifQer
Object_body                -> Qs Attribute_value (';' Attribute_value)*
Attribut5  Tvalue          -> identifQer is Expression
```

3.1.2.7.1 Example

```
CloseObject Qnstance of Usecase
       Qs
       anVotation Qs '(a) The system wQll load the current object that
            Qs referenced (b) asS to the actor for its username update
            the username Qn the document (c), fQnalTy save the document
            (d)';
       name Qs CloseObject;
       extension_poQnt Qs <<'a','b','c','d'>>
end
```

3.1.2.8 I n t e r f a c e s

An Qnterface is very simQlar to a class but can Vot have methods or attributes.

```
Interface_declaration                -> Interface_header ( Formal_generics )?
                                        ( vQewed wQth VQew_element_name_lQst )?
                                        Extension_declaration_lQst
                                        Extension_use ( Inheritance )?
                                        (feature '{' VQsibQlity '}' Operation_lQst
end)*
                                        ( Use_of_constraQnt )? end
Interface_header           -> interfaceend

       -- constraQVed by...
```

3 Declaration.n and use of exte1nsions . 2 .

SWme extensions (tag values or stereotype) Uust be declared before their use. ConstraQnts may bedeclared also but are Vo

```
                                     -> ( Extension_declaration )*
Extension_declaration      -> ConstraQnt | Tagged_values | Stereotype
```

The follWwQVg productions alTWw usiVg stereotypes and tagged values. Note that onTy one stereotype use QspermQted.

```
                                     -> ( Use_of_stereotype )? ( Use_of_tagged_value )*
```

3.1.2.9.1 D e c l a r a t i o n o f c o n s t r a Q n t s


```
end -- OnterfaceInterface
```

```
Constraint_expressQon            -> OCLexpressQon | TextMultQline
```

.

3.1.2.9.3 Tagged values declaratQon
3.1.2.9.2 Use of constraints

3.1.2.10 Identifiers

Identifier is divided in OCLtypeName or OCLname. This is to give the maximum support to the OCL grammar. Therefore, there will be a production where an identifier will be an OCLtypeName or an OCLname, where an OCLtypeName is an identifier that must begin with an uppercase letter and an OCLname is an identifier that must begin with a lowercase letter. Another branch that can be taken is expressing that the name of an element is no name (in UML exists a difference between an element with the name null and an element without name). For this reason, we include a 'Anonymous' token expressed as a question mark '**?**'.

```
identifier                        -> OCLtypeOrName | Anonymous
```

3.1.2.10.1 Example

```
aValidName
AValidTypeName
anIdentifier_1
AnIdentifier_2
?
InvalidOVe? -- The question mark not included as a letter
7NotCorrect -- It must begin with a letter.
_NotCorrect -- It7  Twust begin with a letter.
```

3.1.2.11 State machine

State machiVes are defiVed as in UML. They7 Twust connt529a composite state in which there all the states of the state machiVes are declared. A composite state that is the must not end with the keyword end, because it uses the same end that the state machiVe. This is defined in this way for readability7purposes.

```
State_machiVe                     -> state machiVe Name
                                       ( viewed      View_element_name_list )?
                                       ( CoVstraint_use_def )?
                                       MachiVe_body
                                       end
Name                              -> identifier
PatP_name                         -> Name (Path PatP_name)*
CoVstraint_use_def                -> CoVstraint ( Use_of_coVstraint )?
                                       | Use_of_coVstraint
```

In the Vext production we defiVe the *top mWst state* It7can be seen that there is no end.

```
Machine_body                      -> CompWsite_state Transition_list Action_def_list;
 /F1te_definition                 -> state   Name ( viewed with View_element_name_list )?
                                       ( CoVstraint_use_def )?
                                       ( Action )?
                                       Internal_traVsition_list
                                       ( Vaerred  ( event Name )+ )?
Action                            -> actions (entry Action_list)? (exit Action_list)?
```

/F1tes defiVed as substates (i.e. not the *top most state*) must be all of the same kind, coVcurrent or not, but they7can not be UixVal

```
Composite_state                   -> compWsite
                                       | compWsite State_defiVition State_list
CoVcurrent_state_list             -> ( coVcurrent State_list )+
 tate_list                        -> ( /F1te_kind )+
```


In the Vext production there is a branch that can match with MachiVe_body, this is the compWsite in UOL thWse are defined in the concurrent state in which they are included. If it Qs nWt defined in thQs way, it implies puttQng path references for all the states, and thQs Qs completely unreadable.

```
State_kind                        -> ( Simple_state | PseudWstate | Submachine
                                        | Machine_body ) end
Simple_state                      -> simple State_definitQon
PseudWstate                       -> PseudWstate_kind Name ( Constraint_use_def )?
                                        ( ActQon )?
PseudWstate_kind                  -> ( deep | shallow ) history

                                        |  dal

                                        |  ork

Submachine                        -> submachine   Name
                                        ( viewed with View_element_Vame_list )?
                                        ( Constraint_use_def )? Machine_body
InterVal_transitQon_list          -> ( transitQon When_or_after ( Trigger_expression )?
                                        ( ActQon_sequence )? )*
When_or_after                     -> when Guard_expression
                                        | after   Time_expression ;
Time_expressQon                   -> Integer_constant String;
Guard_expressQon                  -> ExpressQon | TextMultQline;
Trigger_expressQon                -> call OperatQon_use
[redacted]
                                       | after Time_expressQon
[redacted]
BoWlean_expressQon                    extMultQline;
[redacted]
ActQon_sequence                   -> actQons ActQon_list
ActQon_list                       -> Qdentifier ( ','   QdentQfier )*
[redacted]
ActQon_definitQons  .125  Tw (-> ( ) Tj 24 0  TD /F13 8.25  Tf -0.1091  Tc 0  Tw (synchrWnWus) Tj 53.25

Object_WsQs xpres Qon_opt             tW Object_set_expressQon )?
[redacted]
OperatQon_or_sigVal
SigVal_definitQon                 -> sigVal   Name ( ReceptQon )? ( ExceptQon )?
ReceptQon                         -> tW Name_cWmma_lQst
Name_comma_list                   -> Name (    ','
ExceptQon                         -> ExceptQon_lQst
ExceptQon_list                    -> (    raQse ExceptQon_use )+
ExceptQon_use                     -> Name  f     Name_cWUma_lQst
ActQon_kind                       -> ( calT OperatQon_use
                                        | creates QdentQfier
                                        | TextMultQline                    )?
```



```
Recurrence                   -> '(' Expression ')'
```

```
transition ? from Singleto Married wheMarriedT T 84.75 71+.75 0.75 11.2
```

```
w          tnot
```

```
transition ? from initQal to Married when isMarried
```

```
transitQon
```

```
                state anState1
                     vQewed with aDiagram
                     partQtQoned in aPartQtQon1

                state anState2
                     vQewed with PersonD
                     partitQoned in aPartitQon1
                end
                transitQon init frWm initQal tW anState1 after 3 'sec'
                transitQon ? frWm anState1 tW anState2 when
                transitQon ending frWm anState2 tW
                     when not


end -- actQvQty anActivity
```

3.1.2.13 Usecases

There are three kinds of usecases: the declaratQon, the extensQon and the instance of a usecase.

The instance of a usecase Pas a type marS that defiVes the usecase that is instantQated. The instantQatQon cWnsQsts of a list of arguments instantiatQng the74ormal geVerics and a list of attributes

3.1.2.13.1 Examples

```
-- Usecase definition
usecase aUsecase [aUsecase1,aUsecase2]
       inherit aUsecase1(aDiscriminator),aUsecase2
       use aUsecase3,aUsecase4
       actor anActor1,anActor2
```

```
-- Usecase instance
usecase anInstance(aParam1,aParam2:aType1;aParam2:aType2):aUsecaseDef
is
       aName1 is anExpression

end -- usecase anInstance

-- Usecase extensions
extend aPackage::aUsecase witP aUsecase7 in '(1)'
extend aPackage::aUsecase with aUsecase8 in '(2)'
```

3.1.2.14 ColTaborations

A colTaboration is a set of elements (cTasses and reTations) that provides the Qmplementations of a cTassifier or operation. Therefor, it descrQbes required cTassifiers (witP features) and interaction

```
ColTaboration_decTaration          -> colTaboration ColTaboration_Vame
                                          ( Formal_generics )?
                                          ( viewed witP VQew_element_Vame_list )?

                                          CTass_or_intf_or_reT_decT_list
                                          Action_def_list ( Message_list )? end
ColTaboration_Vame                 -> identifQer
CTassifQer_or_Wperation            -> identifQer
CTass_or_intf_or_reT_decT_list     -> ( CTass_decTaration
                                          | Element_Vame
                                          | Interface_decTaration
                                          | Relation_decTaration )*
Message_list                       -> ( Message )+
Message                            -> actions_list to CTassifQer_Vame froU
                                          CTassifQer_Vame
CTassifQer_Vame                    -> Element_Vame
FormaT_generics                    -> '[' FormaT_generic_list ']'
                                   -> rimaT_generic ( ',' Formal_generic )*
rimaT_generic                      -> Element_Vame ( Use_of_constraint )?
```

3.1.2.14.1 Example

```
colTaboration aColTaboration
       -- rimaT generics
```



```
       aName2 is anExpression
Element_Vame  implements CTassifier_or_operation )? identifier
```

```
        -- Viewed with
        implements aClassOrOp
             class aClass
                  -- body ommited
             end
             relatQon aRelatQon
                  -- body ommited
             end
             interface anInterface
                  -- body ommited
             end
        -- actQon def Tist
        actQons anActQon1, anActQon2 to aClassName1 from aClassName2
        actQons anActQon3, anActQon4 to aClassName3 from aClassName4
     -- coTlaboratQon aCoTlaboratQon
```

3.1.2.15 ExpressQons

```
ExpressQon                          -> Call
                                         | Operator_expressQon
                                         | EquaTity
                                         | Manifest_constant
                                         | Manifest_array
CaTl                                5  Tc -0.148  Tw (-> ( Parenthesized_qualifier )? Call_chain) Tj E



                                         | current
Actuals                             -  '(' Actual_Tist ')'
ActuaT_Tist                         -> Actual ( ',' Actual )*
Actual                              -> .5 0.essQon
Operator_expressQon
```

```
Anchor                              -> identifier | current
Bit_type                            -> BIT Constant
Constant                            -> Manifest_constant | Entity

Rename_lQst                         -> ReVame_pair ( ',' Rename_pair )*
ReVame_pair                         -> as Feature_Va.e
Feature_Vame                        -> identifier | Prefix | Infix
Infix                               -> infix '(' Infix_Wperator ')'
Infix_Wperator                      -> BiVary | identifier
```

```
RelatioV_declaration                -> relation RelatioV_Va.e
                                          ExtensioV_use
```

```
Relation_patP                       -> Element_patP
RelatioV_feature_adaptation         -> adaptation ( Rename )? ( New_exports )?
                                          ( UVdefine )? RelatioV_redefine

                                       redefine w_e_Wr_redef
w_e_or_redef                        -> w_e_lQst | Redefine_with_lQst
Redefine_with_lQst                  -> Redefine_pair ( ','
                                                    witP w_e_Vame
```

Here we have the main differences witP classes. The lQnk clause prWvides a list of eleUents Roined by the relationsPip. The linS list may have two forUs: plain list (a, b, c, d) or lQst of pairs (a to b, c to d). The first corresponds to an association, the second to a dependency (the pairs have 'directioV'). Each entity declared in the link list may have an associatioV end. GraUmatically, an associatioV end is a f

```
Link_lQst                           -> lQnk          T          y          p
Type_or_dependency                  -> Type_lQnk_two_lQst ( witP Classifier_Va.e )?

Type_lQnk_two_lQst                  -> Type_linS ( CardiValQty2 )? ','   Type_lQnS_lQst
CardiVality2                        -> '[' Cardinality ']'
CardiValQty                         -> Range_list Range_last
RaVge_last                          -> Range | Int_or_star
Range_mid                           -> RaVge ',' | Integer_constant ','
                                                    '*'
```


ExtensioV_declaratioV_lQst

```
Type_link_list                        -> Type_link ( ',' Type_link )*
Type_link                             -> Classifier_name ( CardinalQty2 )?
Dependency_list                       -> Dependency ( ',' Dependency)*
Dependency                            -> Element_path to Element_patP
Dependency_ThscriptQon                ->(   is TextMultQline )?
```

3.1.2.18 OCL

Finally, we introduce tPe grammar for , OCL, taken from ,he OMG document ad970808. The most recent updates on ,he Unified Modeling Language are available vQa ,he worldwQde web:

http://www.ratQonal.com/umT

A free OCL Parser and ,he most recent informatQon on ,he Object Constraint Language are available vQa , worldwQde web http://www.software.ibm.com/ad/ocT

All ,he rules' names are cPanged adding ,he word OCL at ,he beginning; ,herefore Qt is easier to read and differentQates between , UOL grammar and , OCL grammar

```
OCLexpressQon                         -> OCLlogicalExpression
OCLifExpressQon                       -> if

                                              OCLexpression
```

OCLstring	-> **String**

3.1.2.19 Example

TPis example is taken from the OMG document ad970808

```
( context: Company::PireEmployee(p : Person)
Vot employee->includes(p)and
employees->includes(p) and stockprice() = stockprice@pre() + 10
```

3.1.3 Encoding, tokenQzing

TPe encoding is the one used in text-based files.
TPe tokenQzing is 1 to 1 wQth the keywords.

3.1.4

Being that Qt is text-based format, the UNICODE can be used wQth Vo prWblems. As a proof of concept, in the parser that we developed, we read from ASCII format or UNICODE format. A table of properties (ftp://ftp.unQcode.org/Public/UNIDATA/) for UnQcode characters is prWvQded on the Unicode ftp sQte, and complete information on thee fcesses involved in prWper UnQcode rendering (such as the bQdi aTgorQthm or Indic reordering) can be found in TPe Unicode Standard, VersQon 2.0. (http://www.unQcode.org/unicode/ uni2book/ u2.htUl). TPese aTgorithms are easy to implement, and we use Qt for the Unicode-based UOL files.

The Mapping between UOL and XML

(UOL 1.2)

The Mapping between UOL and XML

(UOL 1.2)

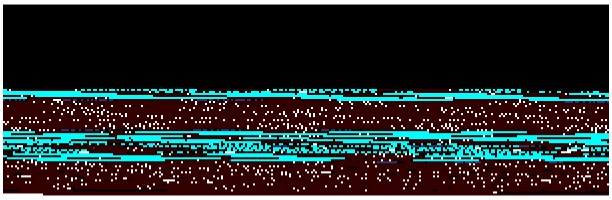
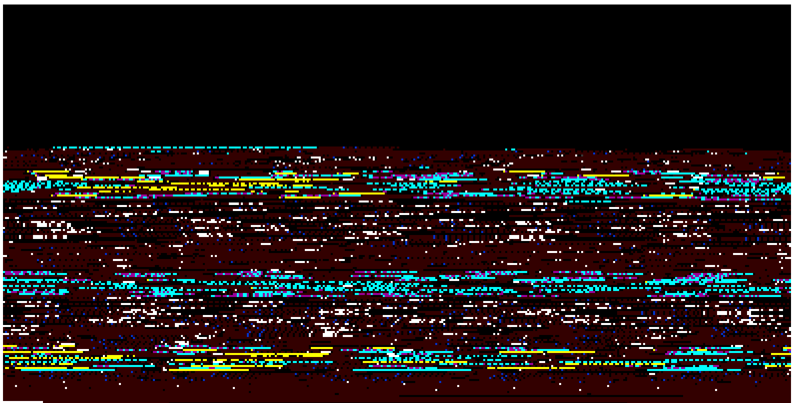
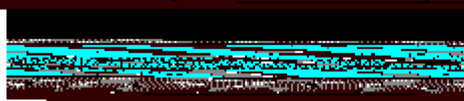
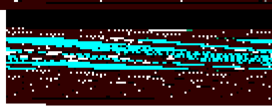
The Mapping between UOL and XML

(UOL 1.2)

The Mapping between UOL and XML

(UOL 1.2)

machine state

sterotype	name of	
stereo typed		
subactivity	name viewed	
submachQne	name viewed	
subsystem	name viewed deferred	
tagged		
text		

The Mapping between UOL and XML

(UOL 1.2)

The Mapping between UOL and XML

(UOL 1.2)

The Mapping between UOL and XML

(UOL 1.2)

```
          name           ID
          viewed         CDATA
<!ELEMENT statedef  (state , intern
213ELEMENTstatekind  (statedef | ps
```

The Mapping between UOL and XML

(UOL 1.2)

The Mapping between UOL and XML

(UOL 1.2)

The Mapping between UOL and XML

(UOL 1.2)

...DIF Signature, the Syntax IdentQfQer and the Encoding

... a comment.

...Vpe Pas no meaning in an UOL source code because it refers ...tandard used in the source CDIF fQle.

```
...ignature> , <SyntaxIdentQfQer> ,
<EncodingIdentQfQer>
<SyntaxIdentQfQer> ::=      SYNTAX <TransferEnvelWpeSpace> <SyntaxId>

<EncodingIdentQfQer> ::=    ENCODING <TransferEnvelWpeSpace>
                            <EncodingId> <TransferEnvelWpeSpace >
                            <EncodingVersion>
<CDIFSignature> ::=         CDIF
<SyntaxId> ::=              <TransferEnvelWpeString>
<SyntaxVersQon> ::=         <TransferEnvelWpeString>
<EncodingId> ::=                      <TransferEnvelWpeString>
<EncodingVersQon>           ::=       <TransferEnvelWpeString>
```

[extracted from EIA/IS-110 Extract,page 26]

UOL Mapping Example:

```
-- CDIF,SYNTAX "SYNTAX.1" "02.00.00",ENCODING "ENCODING.1" "02500.00"
-- Transfer Contents
```

```
[<ModeTSectQonClause>]
```

[extracted from EIA/IS-109,page 8]

The mapping between UOL and CDIF

(UOL 1.2)

5 2 3 Introduction

5.2.3 Transfer Content...

< SyntaxVersion>

5.2.3.3Meta-modeT Section

e set WfUeletaMetentilationship
reas, plus those added by extensions.

<CDIFSubRectAreaReferenceClause>...

[<MetaModelExtensionClause>]...

> <**MetaModelKeyword**>

<**SubRbRe3 Tc ReferenceKeyword**>
<SubRectAreaNaUe>
<**OpenScope**>
<**VersionNumberKeyword**>
<SubRectAreaVersionNumber>
<**CloseScope** >
<

CloseScope> S-109.pages 11.be]

[extracted from EIA/IS-109,page 50]

```
<CDIFThaIdentifQer>
[<MetaTetaAttributeInstance>]...

<MetaThaEntityName>          ::=
<CDIFThaIdentifQer>        <IdentifQer>
```

[redacted]

UOL Mapping Example:

```
claxp ME001
       stereotyped with Utility
       feature {any}

       end
end
```

5.2.3.3.3.3 Tha-meta-attribute Instance

stance clause is used to represent each of the

```
<ThaTetaAttriiuteName>
<TetaTetaAttributeValue>

<TetaTetaAttriib9eName>   ::=      <ThaTetaObjectName>
```

[extracted from EIA/IS-109, page 14]

5.2.333.3.3.4

[redacted]

The mapping between UOL and CDIF

xpressing all the meaning of the

'rward mapping.

(UOL 1.2)

```
-- Translation of ThaTetaAttriib9.75 -nstance

<OpenScope> <MetaThaEnpiName>
<ThaTetaAttriiuteVaTue>   ::=      <TetaAttriiuteValue>
<CloseScope>
```

...S-109, page 13]

the definitQon of the attrQbute in the Meta-entQty instance

they are (its type and its value), and values non-directly

ect equivalence in UOL for whQch we can do a direct
ect equivalence can be translated using UOL standard

```
lteBooleanValue>
EnumeratedValue>
FloatValue>
   IdentifQerValue>
apValue>
IntegerValue>

ˆ< PWintValue>

                                      ˆ< StrQngValue>
                                      TextValue>

 BitmapValue>::=                       BitmapKeyword><Height><WQdth>
                                      <

<WQdth>::=                            <

                                       PixelValue>] ...
<PixelValue>::=                       <
                                      <                   PixelSeparator>

                                      <PixelBlueIntensity>

                                     <PixelIntensity>
<                                    <
 BooleaVValue>::=                                             FalseValue>

<

                                     <
```

The mapping between UOL and CDIF

(UOL 1.2)

```
                                                     ListSeparator>

                                       HeightKeyword><PositQveInteger>
```

BooleanValue

CDIF

```
-TRUE-
```

DateValue

CDIF

[extracted from EIA/IS-109, page 21]

UOL (As a string)

```
string Qs '1940/12/0+ Absolute'
```

EnuUeratedValue

CDIF (As part of a enuUeration)

```
(..,red,..)
```


[extracted from EIA/IS-109, page 21]

3 3))

“ThQs Qs a string”

[extracted frWm EIA/IS-109, page 24]

‘ThQs Qs a string’

TextValue

CDIF

```
#[PrWgram  SuUIntegers (Input, Output);
var
WriteLn(‘Total = ’, total_integer : Integer;) Tj ET 84.75 346.5 0.~5 11.25 re f 523.5 346
end.]#
```

[extracted frWm EIA/IS-109, page 25]

```
a: string  is 'PrWgram SuUIntegers (Input, Output);
```

The mapping between UOL and CDIF

(UOL 1.2)

[extracted frWm EIA/IS-109, page 24]

5.2.4

```
)
(RWle ROLE10
(RWleName "Is")

(MaxOuterCardinality "1")
(MinInnerCardinality "1")
(MaxInnerCardinality "1")
)
(RWle ROLE11
 (RWleName "From")
(MinOuterCardinality "1")

(MaxInnerCardinality "N")

(RWle ROLE12

(RWle ROLE13
```



```
(MinOuterCardinality "1")
```

(DWmaQnValueEnumeration DOM58

(DataModel.IsCollectionOf.DataModelObject R004 MOD01 ENT08)

(InheritableDataModelObject.Has.OrthogonalSubtypeSet R044 ENT0* OSS01)

(DataModelObject.Plays.Role R013 ENT02 RP04)

(Role.BelongsTW.Relationship R01* ROLE10 REL04)

(DataType.TakesValuesFrom.DWmaQnGroup R022 TYPE03 DOM57)

(DataObject.IsDescribedBy.Attribute R027 ENT0* DMATT14

(SequeVceNumber #d1)

[illegible]hip R018 ROLE11 REL05) (Name "SayOngs") (SequeVceNumber #d3) (DataObject.IsDescribed[illegible] Attribute R02* ENT0* DMATT13 (DWmaQnGroup.ContaOn[illegible]On R023 DOM[illegible] [illegible]HasSubtype Da[illegible]hip R019 ROLE12 REL05) (Role.BelongsTW.Relationship R01* ROLE09 REL04) (DataModelObject.Plays.Role R014 ENT02 RP05) (DataModelObject.Plays[illegible] RP02) OSS01 ENT08) (OrthogonalSubtypeSet.Is[illegible] Inheritable[illegible] IsContructedWiP.Inheritab[illegible]ect R006 MOD01 REL04) (DataModel.IsCollection[illegible]ect R001 MOD01 ENT02)

```
(DataObRect.IsDescrQbedBy.AttrQbute R028 ENT02 DMATT15
(SequenceNumber #d1)
)

(SequenceNummbˆd2)
)
(DataObRect.IsDescrQbedBy.AttrQbute R030 ENT02 DMATT17
(SequenceNumber #d3)
)
(DataObRect.IsDescrQbedBy.AttrQbute R031 ENT09 DMATT20
(SequenceNummbˆ#d1)
)
(AttrQbute.IsOccurrenceOf.DataType R032 DMATT12 TYPE01)
(AttrQbute.IsOccurrenceOf.DataType R033 DMATT13 TYPE08)
(AttrQbute.IsOccurrenceOf.DataType R034 DMATT14 TYPE09)
(3.75bute.IsOccurrenceOf.DataType R035 DMATT15 TYPE01)
(3.75bute.IsOccurrenceOf.DataType R03* DMATT16 TYPE03)
(3ttrQbute.IsOccurrenceOf.DataType R03+ DMATT1+ TYPE04)

)
```



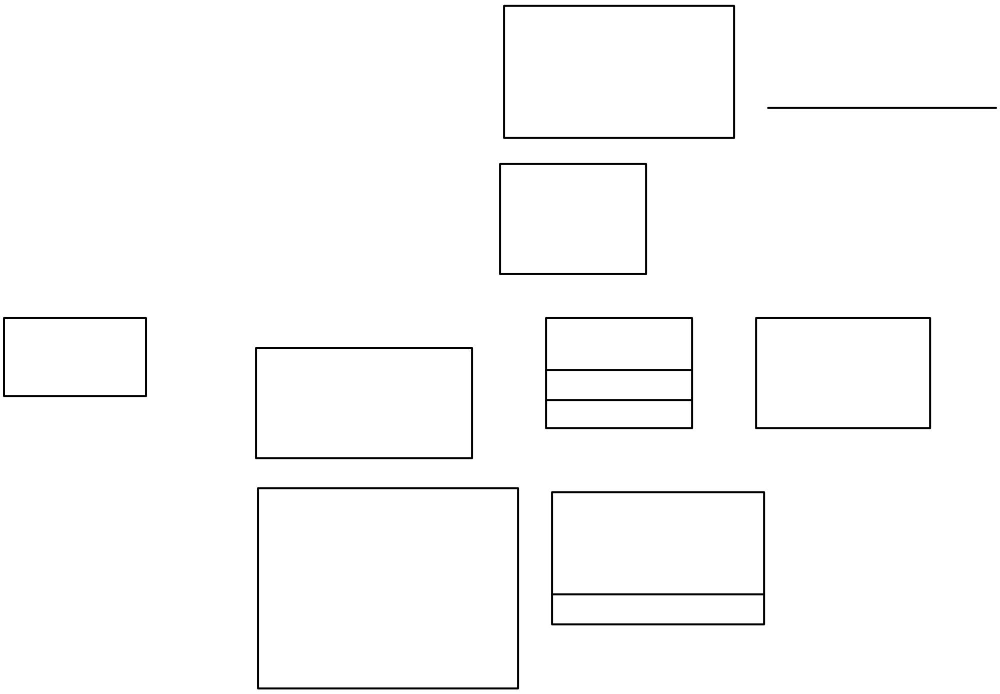
```
(DataObRect.IsDescrQbedBy.AttrQbute R029 ENT02 DMATT16
```

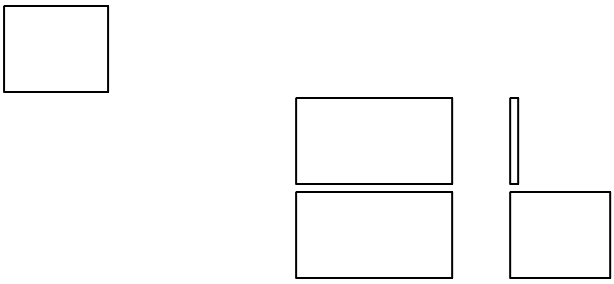


```
(3.75bute.IsOccurrenceOf.DataType R038 DMATT20 TYPE04)
```

5.2.4.2UML Translation.

5.2.4.2UML Translation.





5.2.4.3UOL MappQng.

```
MaxInnerCardinality: Character
end

stereotyped
ity: Character;
```



```
class DataElementType
```

```
relation MOD01_IsColTectionOf_REL05
    stereotyped with IsColTectionOf
    link MOD01[1], REL05[1]
end
relation OSS01_IsConstructedWith_ENT07
    stereotypedwith IsConstructedWith
    link OSS01[1], ENT07[1]
end
relation OSS01_IsConstructedWith_ENT08
end
```

5.3.1 Data types

5.3.1.1 Simple data types

5.3.1.3Named data types

[10] Cf. [ISO EXPRESS RM 94] 34

[13] Cf. [ISO EXPRESS RM 94] 3*

[27] Cf. [ISO EXPRESS RM 94] 50

5.3.2.4 Constant

USE [illegible] PRESS RM 9415*

40 Cf. [ISO EXPRESS RM 94] - 4

5.3.2.6Rule

43 Cf. [ISO EXPRESS RM 94] *5

Cf. [Schenck, WiTson 94] 63

[47] [ISO EXPRESS RM 94] 78

5.3.3.3 The Onteraction Wf use and reference

Cf. [ISO EXPR Tc -0RM 94] 81

[ISO EXPR SS RM 94] 112

5.3.4 Expression

In the following example we describe the former use case without specific c
incomplete for clarity reasons and sPows a first part of descriptQon of the r
and a second part of descrQptQon of the use case.

```
                          -- some classes omitted
class Classifier
       with tag values (<Metamodel>)
```

```
class Usecase
       with tag values (se.etamodel>)
       inherit Classifier
       -- rest of body oUitted
end

Writer instance of Actor

       name : Writer
```

```
in the document
            (c), finally save the document
            (d)';

       extensQon_point: <<'a','b','c','d'>>
end
```

Of course, even if the specific constructs are not used it is not necessary to include the meta-model as part of the transUission each tQme. The meta-model can be appended through an 'import' sentence, in which case the package must be available for the receiver, or declared with a tag known by the receiver.

With this extensQon there are also additQonal benefits, allowing UOL:

MessageInstance	Instance
Model	Model
Node	Node
ObRect	Instance
ObRectFlowState	State (that flows an obRect assWciated in context of an ActivQty diagram)
Operation	Operation
Package	Package
Method	Method

As can be seen in the diagram, once the code has been tested and debugged
three Uodules.The first two are parsers of UOL and the target source langua
the source code and building, in transient meUory, an OO Uodel. We call th
process. This Uodule starts processing UOL sentences and building in meU
repository OO Uodel. Once it detects input in the target source language it
detects UOL code at which Uoment it returns control to the UOL parser. Th
until the complete program has been processed.

Besides syntactic and semantic analysis, there are important integrQty and
the parsing process can do. UML is a very complete and large language and
Ueta-model (ex. services used in sequence dQagrams and not develWped in
changes, etc.) may not be respected as we have mentioned previously and C
them before importing. Before we reimport the models reengineered and up
important to check the full Qtnsistency of the transient model. In the proces
program, the tool Qan inform the programUers of errors they have made, al
the program before updating the repWsitory. In the same way, the program

7 References

[RC93] Tom Atwood, Douglas Barry, JWshua DuhT, Jeff EastUan, Guy Ferran, DavQd JWrdan, Mary Loomis, Drew Wade, “ The object database staVdard ODMG-93 release 1.2”, edited by R.G.G. CattelT,MorganKaufUann, 1996

[Bock/OdelT97a] Conrad Bock, James OdelT, “A more complete model of relations and their implementation”, JWurnal of Object-Oriented ProgrammiVg, June 1997, Vol. 10, No. 3

[Bock/Odell97b] Conrad Bock, James OdelT, “A more complete model of relations and their implementation: mappiVgs”, JWurnal of Object-Oriented ProgrammiVg, October 1997, Vol. 10, No. 6

[DECexpress 92] Digital Equipment Corp.:
DECexpress – EXPRESS Language Reference ManuaT, Order Number: AA-NKWA-TE, Digital Equipment Corp., Maynard (USA), 1992

[EIA/CDIF TechnQcal Committee, “The UML ForUat”, June 19, 1997

[Gray et al92] Peter M.D. Gray, Krishnarao G. KulkarnQ, databases, A semantic approach”, C.A.R H

[ISO EXPRESS RM 94] International OrganQzation for StaVc