OSL Object Specification Language (proposal)

Zygmunt Ryznar
Cracow, Poland

OSL is a descriptive specification language aimed at improving documentation and communication by providing simple formal description of any object in terms of structure and behaviour. A vocabulary of this language is a small set of simple English words. The precise compact „dictionary" of words and statements is the base upon which one can compare objects independently of country, time, profession, type of business. Author presents the kernel and subsets of OSL covering such different areas as business, human-being and computer program. [1]

Categories and Subject Descriptors: • **System description languages** • **Specification languages**

I. INTRODUCTION

The specification languages are focused usually on the following domains:

a. the collection of entities, their relationships and attributes (ERA - e.g. PSL–Problem Statement Language in ISDOS)
b. the collection of objects, their structure, behaviour, unlimited relations (not restricted to dbms)
c. documenting programs written in a given language (e.g. ANNA – ANNotated Ada)
d. an overall program description (e.g. PSL – Program Specification Library)
e. modelling programs in terms of data types and functions performed upon them (e.g. VDM-SL)
f. a real time process control (e.g. SDL from TUI).

The OSL belongs to the **b** category. Almost all languages are oriented toward information systems and relations embedded in data bases while scope of OSL includes any object (real world, information, program etc.) and innovative data structures (free-space, swarm, network, hierachy, line, triangle, tunnel, curve).

OSL is a descriptive language, so it should not be compared with algebraic languages such as CASL, ASM, ACM. It also has no features (at least in the current version) for mathematical modelling of simulations e.g. calculating time dependent variables such as net income for each month of a year.

It differs from the SDL (Specification and Description Language), which focuses on process control and real-time applications with asynchronous communication between components.

It is not as extended as UML (Unified Modeling Language), VDM-SL (Vienna Development Method – Specification Language) and not equipped with diagram technology.

OSL is a tool for a simple but formal description of any object in terms of structure and behaviour. The vocabulary of this language is a small set of simple English words. The precise compact „dictionary" of words and statements is the base from which one can compare objects independently of country, time, profession, type of business etc.[2]

The kernel of OSL contains common (standard) phrases and keywords, dedicated to any object. An extension of the language for selected areas is defined in subsets called here OSL-B (business), OSL-H (human-being), and OSL-S (systems). This collection of subsets presents possibilities (and limitations) and is extendable for new areas. A subset is similar to a shell in an operating system for a group of users allowing each group to use its own commands and keywords.

The „engine" (to be developed) of the OSL language should be capable to process the contents of the specification at all possible analytical levels, trace relations between objects, and present them in tables and graphs, verify (if possible) and make the final global presentation of a given area. Concerning graphs, it may be desirable to present some features such as a repetition as a simple iteration, single spiral or multiband spiral; the structure (herein called layout) as free-space, swarm, network, hierarchy, line, triangle, tunnel. The precise definition of these figures is beyond the scope of this publication. The spiral provides inspiration for representing the learning processes and learning curve. Others, although new, are intuitively understandble.

Objects may be concrete or abstract (existent or conceptual,virtual) independent or dependent, etc. An object has at least its own name, identifier, properties, structure, interface, behaviour and "history". An important feature of the object is its ability to collaborate with other objects using visible relations and interface. External (outside of the area or subject) links may be specified using definitions in the ENVIRONMENT section.

The "subject" is the main class of an object within area, e.g. „bank" in area of „banking". The subject list may consist of a set of types (e.g. „bank" may be universal, retail, wholesale, short term cash loans etc). The specification and definitions may be common for all banks and private for each type or even for a given bank. The definitions in the kernel are common to any area.

The documentation of an OSL application should contain a kernel (global definitions), area subset definitions and (optionally) a detailed specification worked out for objects which belong to the area. A definition section deals with "abstract objects", while specification contains descriptions of physical objects (instances of abstract objects) using phrases and keywords located in definitions. Definitions written in the specification are local (like local variables in computer programs).

[1] This work was not supported by any organization

[2] *My profession is in the field of information system analysis and design. During the course of many years of interaction with business and IT professionals, I concluded that there is "a must" for a precise specification of subject matter.*

One vital question concerning the implementation of OSL is a computer-aided support to input, correcting the specification (after validation) and outputting readable results. In further development one can imagine an „Application Software Factory”, that generates ready to use application software from definitions, specifications and skeletal building blocks.

II. OSL NOTATION

<!...>	comment
< >	container
=>	link to something external (outside area)
<def ></def>	start-end of language/subset definition
<spec </spec>	start-end of a given object specification
iiiiiiii	keywords: *by,from,to,when*
::=	type assignment
:=	list items
=	value assignment
:	name assignment
@	mark of attibute,feature,property
::	belongs to
(x,y,..)	list
[name]	executive/operational object
xxxxXxxx	special or complex names
XXXX	basic object
UUUU.XXXX	qualified name of object
xxxxxxxx	some complex or important keywords
KKKKKK	OSL keywords ((capital letter)
xxxxxxxx	operational keywords: event,action
<**xxxxxxxx**>	tags: id,def,spec
& /	conjunctions: and or

Writing of specification should be supported by a specialized editor, that automatically (based on the OSL notation) converts words into bold, inverted font and capital/small letters. Otherwise all the text (excluding object names) may be written in lower case. Another useful facility would be virtual keyboard with such keys as <def <spec ::= := :: etc.

Generally, OSL needs a software engineering support like IDE, SDK, graphic tools, object standards (like DCOM, Corba etc.) and database technology.

III. OSL KERNEL DEFINITIONS

```
<def  OSL>
<def  ENVIRONMENT: ENV >
        ENV:=(REGULATIONS, INFRASTRUCTURE)
        ENV.INFRASTRUCTURE:INFR
        ENV.REGULATIONS:=(LegalActs, Resolutions,
                                    Decisions)
       INFR:=(IT,ORG,HR)
       INFR.IT:= (Servers,OperSystems,Applications,
                    TransDataBases,Users, prLanguages)
       INFR.ORG:=(OrgStructureOfCompany)
       INFR.HR:=(HumanResources)
</def>
<def  globalMapping><!made automatically by OSLpackage>
   defLang:=(BSL,HSL,SSL)<! defined subsets of OSL>)
   defList:=(<list of  defined objects>)
   specList:=(<list of  specifications>)
</def>

 <def subject:<NAME><!main object name>
 <def  <name><!other object name>
  object.id<!object identifier>
  class<!class tree>
  object.type::=(eobject<!elementary atomic object >,
                      dobject<!dynamic object >,
                      iobject<!informational object >,
                      vobject <!virtual object >,
                      sobject <!smartobject)>,
                      oobject<!open object>)
@sobject:=(noiceReduction,selfTeach,selfRepair,selfKill,
              selfRestore,selfRestart)
@oobject:=(input(parameters,data),output(info,messages),
              structure(addComponent,addRelations))
dynamics::=(event,operation/transaction,action,process)
dynamics::=(ev,op/tr,ac,pr)<!short notation>
dynamics.scenario::= (scevt,scop,scac,scpr)
event:ev<!-elementary atomic fact >
trans<!transaction in terms of operating system monitor>
ftrans<!financial transaction>
operation:op
action:ac<!sequence of operations or events,long transaction>
process:pr<!sequence of actions and events>
reverseMode::=(rev,rac,rop,rtr)<!back to the previous state>
pr::=(trigger,action(<events>),endEvent)
scenario:sc<!predicted sequence of actions and events>
scenario.rank:=(best,worst)
object.Info<!information visible at the moment of access>
keywords:kwords<!additional keywords in def>
olh<!object life history>:=(timeline,events,aging-curve)
object .role:=(interface, trigger,generator,agent,integrator,
                component,monitor,commander,
                executor/performer,initiator,terminator,
                destructor,participator,owner, stockholder,
                customer,supplier;partner,employee)
relations::=(activatedby,activates, assisted by,
              appearence depends on , belongs to
              /is owned by , built from ,
              calls <obiekt> (<interface>),
              consists of <parts>,contained in/contains,
              controlled by/controls,derived from,
              existence depends on,exists when/in/for,
              included in,linked to ...by/links,
              refers to, relates to, related by affinity,
              represented by/represents,involved in,
              shared by/shares, used by/uses)
 state:=(active,inactive,dormant,suspended,aborted,
          idle/waiting,lost,expected,deleted,homeless)
 status:=(generic,real,virtual,undefined)
 role:=(driver,trigger,reactor,agent,executor,generator)
 reactor::=(acceptance, rejection,constructor)
rank:=(critical,necessary,most wanted,optional,worst,best)
 rule:=(decision-table,logical-when-if,formula).
{control-flow
 ac::=(ev1,ev2,ev3, ..)
 pr::=(ac1,ac2,ac3,...)
 s(ev1,ev2,ev3, ..)<!sequential flow of events>
```

p(ev1,ev2,ev3, ..)<!parallel flow of events>
pr::=**s**(ac1,**s**(ev1,ev2,ev3),ac2(**p**(ev4,ev5,ev6),(ev7,ev8,..))
    <!example of mixed flow>
repetition:=(iteration,single spiral,multiband spiral)
  <!spiral pattern may be used as a pattern of learning -
    each scroll of spiral can be different>
activated *by* <..> *with* <initial-value> *at* <time-point>
    when <condition>
finished *at* < > *with* <...> *when* .<..>}

<**def body**>
  {**body**::=(Contents,Script,Metadata)
    contents<!e.g. document contents, program code >
    script<!operation script generated upon the pattern
        of behaviour >
    metadata<!body structure description >}
    layout:=(free-space,swarm/hive,network,hierachy,line,
        triangle,tunnel,curve)
  <**def**>

</**def**>

IV. OSL-B OSL FOR BUSINESS

OSL-B named BSL (Business Specification Language) is a subset of OSL (Object Specification Language) dedicated to business objects. The simple banking simple example presented below is for illustration purposes only. The full specification should contain also such objects as headoffice, branch, channel of product delivery, customer, deposit account, loan accout, loan credit line and executive operational objects, like account manager, teller, dealer and IT infrastructure objects.

<**def** OSL-B:**BSL**>
BUSINESS:=(BANKING,MANUFACTURING, SERVICES)
ENV.business **id**:=BIC<!Business/Bank Identification Code>
<**def subject:**BANKING >
BANKING:=(RETAIL,WHOSALE,UNIVERSAL,MONEY-MARKET,DERIVATES,SHARES)
ENV:=(bank.**id**:=BIC,account.**id**:=IBAN<!International
    Bank Account Number>,dadaTables)
dataTables:= (LIBOR,OperatingCurrences,ExchangeRates)
kwords:=(customer.**id**,accountCurrency,creationDate,
  ftransLimit,infoSet,Resources,
  operation<!e.g. monthly charge>
  action<!complex activity e.g. defining provision for
    doubtful loans >
  driven/sorted by (ftrans,product, customer, date,
      schedule,frequency)
  matched/matches<!e.g. confirmation>)

<**def** BANKING.RETAIL>
retail.**product**:=(CURR-ACCOUNT,DEPOSIT,LOAN)
<**def subject:**BANK>
object.**Info**:=(BIC,country, bCurrency<!base currency>,
    FinancialYear, number of branches>)
dataTables:=(CorrespondentBanks,Branches,
    calendarWorkingDays,bkAccountChart,
    productList,interestRateTable)
kwords:=(branchNr,customerId,accountNr,rate,
    balance,balanceSheet)
OperationalObjects:=[teller,accountMgr,customerMgr,
    productMgr,trader]
Bank.objects:=(product,currency,limit,account)
limit:=(country,industry,customer,currency)
iobject:=(customerPosition,monthlyBalancesheet)
bank.**type**:=(dmBank<!domestic bank>,
    frBank<!foreign bank>,
    corrBank<!correspondent >)
    bkAccount:=(bsAccount<!balance-sheet>
    nbsAccount<!nonbalance-sheetAccount >
batch-operations:=(eodoperation<!at end of day >,
    eomoperation<!at end of month>,
    eoyoperation<**!**at end of year>,
    eoppoperation<!at end of product>)
  <**def** Account><!customer Account>
    object**Info**:=(Account id,owner,co-owner)
    minBalance,actualBalance,historyStatement)
    Relates to Customer**id**
    rttrans:=(Open,Quit,Cash-in,Cash-out,
      transfer)<!real time transaction>
    eomoperation:=(printMonthlyStatement)
  </**def**>
</**def subject**>
</**def** Banking.RETAIL>
</**def** BSL>
<**spec** Banking.RETAIL(IndustryBank)
  BIC=ALBPXLPW
  customer.**id=XXXXXXX**
  current.account.**id**=PL 99 9999 9999 9999 9999
  owner=John Stale
  co-owner=Jane Stale
  baseAccount.currency =USD
  creation.date=.<...>
  transaction-limit=<...>
  info.set :=(ftrans.incoming,
      ftrans.outcoming,balance)
      @sorted by date
  @:=(deposit,loan,multibranch,echannels)
</**spec**>

V. OSL-H OSL FOR HUMAN

OSL-H named HSL (Human Specification Language) is a semiformal notation for a human being for anyone interested in ontology and existential psychology. The topic "human being" is not a simple one. Several psychologists and writers have stated that the several major theories on personality are colored by subjective factors and motivations affecting each theorist as an individual. Some examples of the diversity of theories are "Functional autonomy" (Allport), "Basic concepts for a psychology of personality" (Murray' H.A),"Trait theory", "16 Personality Factors", "9 Enneagram types" and "Myers-Briggs Type Indicator". Further development of HSL could be achieved with collaboration of psychologists and medical professionals.

One possible usage of HSL language is creation of a human resources database in a corporation or even on an international scale for locating individuals which meet certain psychological, intelectual and professional requirements.

Similarly to other subsets of OSL this one contains only additional phrases and keywords that do not exist in kernel of the base language.

```
 <def OSL-H:HSL>
<def subject:HUMAN>
        class1:=(animals.mammalia.primates.homidae)
        class2:=(nation.ethnic-group.profession.person)
        kwords:=(life-space,behaviour,scope )
        scope::= (BIOPHYSICAL,GEOGR,CULTURAL,
                  SOCIAL, LEGAL)
<def ENV>
       ENV:=(WORLD,CONTINENT,COUNTRY,
                REGION,SITE)
       ENV.LEGAL:=<!Legal acts, resolutions, decisions>
       ENV.CULTURAL:=(tradition, history, education,
                          religion, ideology, art, radio-tv)
       ENV.BIOPHYSICAL:= (animals.homosapiens)
       ENV.GEOGR:= (homeaddress,company,school)
</def>
 <def  PERSON>
       object.Info:=(id,sex,birth-data)
       homeaddress::= (country,site,street,house,flat)
       sex=(male/female/x )
       family::=(gentree,parent,child,son,daughter,
               grandSon,grandDaughter,granMa,granPa)
        emotion:=(love,hate,satisfaction,frustration,
                     agression,enjoyment,anger)
        psychComplex:=(fear of insupport,regression,
                        inferiority,persecution)
        habit,hobby,profession,
        health:=(measures, physical-examinations, illness-
                history),
        role::=(advisor, spouse,
             manager,patron,partner,customer,
supervisor,participator,
             owner,supplier),


        appearence depends on ,
        assisted by,belongs to,
        matched/matches<!e.g. marriage >
        relates to <family-members>
        used by,uses,not used,misused,abused
        state:=(active,inactive,dormant,suspended,
               aborted,idle,lost,dead,homeless,retired,
                married/divorced/single,ignored)
        place:=(point, area,everywhere,nowhere>
        life-space:=(psychological,social,educational,
                     financial)
       behaviour<!flow of processes of the object >
       behaviour.rational::=(selfrealization,need,
                               satisfaction)
 behaviour:=(marriage,friendship,career,ilness,aging)
 genotype,fenotype,
 olh<!object-life-history:=[birth,aging-curve,
         social_events,health_illness-events, educ-events,
         job-events, critical_events, death ]
cluster<!GlobalFactor- estimated on the base of several
         particular factors >
cluster:= (self, profile/type, attitude,leadership, ability,
           extraversion,anxiety,independence, healthState,
           lifeStyle,creativePotential,happiness,
           BipolarPersonality)
self:=(self-identity,self-assesment,self-sentiment,self-
           esteem,self-regard,self-reliance,self-control,
           self-image,self-extension,self-structure)
leadership:=(assertive,creative,facilitative,independent,
                stable,permissive,leadershipStyle,
                leadershipPotential)
ability:=(toughMinded/openMinded,creative,fast/slow,
              toleratesDisorder/perfectionistic,
             grounded/abstracted, improving own learning,
             problem solving, IQ, ......)
need:=(biological(food,medical,emergency,rescue,
           coping),cultural,psychological(love,esteem,
          selfrealization),financial-security)
BipolarPersonality:=(Warmth,Reasoning,EmotionalStability
             Concillation,Dominance,Liveliness,Openness,
             Tension,Rule-Consciousness,SocialBoldness,
              Sensitivity,Vigilance,Abstractedness,
              Privateness,Apprehension,OpennessToChange,
              Self-Reliance,Perfectionism)
```

Warmth=Reserved/Warm
Reasoning=Concrete/Abstract
EmotionalStability=emotional/stable
Concillation=concillatory/aggressive
Dominance =Deferential/Dominant
Liveliness =Serious/Lively
Openness=extraversive/introversive
Tension=Relaxed/Tense
Rule-Consciousness=Expedient/Rule-Conscious
SocialBoldness=Shy/Socially Bold
Sensitivity=Utilitarian/Sensitive
Vigilance=Trusting/Vigilant
Abstractedness=Grounded/Abstracted
Privateness=Forthright/Private
Apprehension=Self-Assured/Apprehensive
OpennessToChange=Traditional/Open to Change
Self-Reliance=Group-Oriented/Self-Reliant
Perfectionism=Tolerates Disorder/Perfectionistic

```
</def >
 </def  HSL>

<spec HUMAN(John Example)>
        sex=male
        family=(married, parent of 3, grandfather of 4)
        state:=(retired, active)
        cluster.self=average
        ability=(openMinded,creative,fast,abstracted)
        need=psychological.self-actualization
        temperament=(emotional,sensitive,
                       introversive,tense,reserved)
        cluster.happiness=good+
        IQ=> http://www.iq-test.com/
<\spec>
```

VI. OSL-S FOR SYSTEMS, PROGRAMS AND APPLICATIONS

Subset OSL-S named SSL (System Specification Language) may help the structured design methodology dealing not only with “well” (usually it means hierarchically) structured problems and systems built under a long-term schedule of integration. A real cause for computerization should be an actual business need (there must be a manager who wants information for decisions). The basis for such a structured design is an understanding the business of company and afterwards the creation of initial state of the system, called the pre-system, containing fundamental technological elements built according to the rules that enable their use in many applications and in changeable environment. There is a need to develop a specific technique to help programmers design skeletal programs that have standardized (but sometimes multifunctional) control flow and are equipped with many “modifiers” that require processing by a tuner. The tuner may perform operations such as inserting data names and parameter values, choosing entry points, generating CALL statements, generating empty modules (driver or stub type), inserting expressions into macrostatements and invoking database schema.

```
<def OSL-S:SSL>
INFR.IT:= (servers,operSystems,applications,
                 transDataBases,dataWarehouses,
                 networkMgtSystem,users,prLanguages)
<NAME>::=(SYSTEM.SUBSYSTEM.MODULE.
             PROGRAM.PR-BLOCK)<!structured name>
 PR-BLOCK<!building block,generic program/subprogram>
 kwords:=(version,interface,run,runTime,integrated,
                 standalone,inDevelopment,accepted,
                 notAccepted,library,creatDate,updDate,tested,
                 rejected,pcode,ecode,callValue,trace,errorCode,
                 inItems,outItems,flow)
trace::=(path,callValue,outValue,errorCode)
process::=(trigger,action(<events>),endEvent)<!when process is
invoked simultaneously by many programs each instance is
recognized by pcode, event by ecode>

<def  pr-name>
       object.Info:=(pr-name,version,author,creatDate,
                          updDate,prLanguage,operSystem)
     { control-flow
          activated  by  <program/procedure-name> with
         <initial-value> at <time-point > when   <condition>
          finished at < time-point /no-of-repetition> with
          <value/output> when <condition>}
<\def>

<def  PR-BLOCK(name)<!reenterable ProcName)>
     objectInfo:=(name,version,author,updDate,codeSize,
                       prLanguage,operSystem)
     resources:=(dataBuffer,eventTrace,stackHandler)
   <def  flow>
      {trigger::=call
        ac(verif)
           ev(checkPassword,callVerif)
               when callVerif  failed exit
        ac(initial) when first call
             ev(bufferDecl,stackDecl),
        act(tuning)
            ev(paramAnalysis,transform,generateExecutable)
        ac(activate)
             ev(paramAnalysis,tuner),
        ac(run)
              ev(load,perform,releaseResources,exit)
          }
      </def>
      <def interface>
        <!at run-time interface retains an actual  state of
             resources  for each call>
           call::=(<callingName>,<calledName>,<tunerName>
                <password><!optional>,<entryPoint>,
               (<parameters,modifiers>),inItems,outItems)
        <\def>
  <\def>
```

Structure of specification:

```
<spec SYSTEM:<name>
        SUBSYSTEMS:=(list of subsystems)
        <spec SUBSYSTEM:<name>
               <spec MODULE:<name>
                      <spec PROGRAM:<name>
                       ......
                      </spec >
               <\spec>
         <\spec>
<\spec>
```

VII. CONCLUSION

OSL is a conceptual schema of the „world” centered on the structure, behaviour and relations of objects. This language is oriented on communication between people and in case of the IT documenting core features of „system” before, during or/and after design.

In reality any business is highly complex and dynamic. The key to meet this challenge is improving a communication between busines and IT people, modernization of design approach to create systems with variable structure and undefined borders using a library of tunable predefined generalized blocks. Future IT systems should function as a collection of many incarnations based on the same generalized set of blocks, which can be located anywhere in the system, tuned and properly interfaced. It would be a good “hybrid” solution both for the specification as well as for execution.

The OSL language equipped with computer-aided tools would seem be a useful initiative to facilitate the documentation of structured design centered on objects, events, processes and structure.

REFERENCES

[1] Zygmunt Ryznar.1978.A conceptual model of an interfunctional data base system. Information and Management 2/1978

[2] Zygmunt Ryznar.1981.S&DL – Structured Design Language.

[3] 526-533. Angewandte Informatik-Applied Informatics.12/1981