TR-99-3

COMPONENT-ORIENTED SOFTWARE ENGINEERING MODELING LANGUAGE: COSEML

Ali H. Doğru

Component Oriented Software Engineering Modeling Language: COSEML

Ali H. Dogru

Computer Engineering Department,

Middle East Technical University

December, 1999

Component Oriented Software Engineering Modeling Language: COSEML

Ali H. Doğru

Computer Engineering Department
Middle East Technical University

Abstract

A graphical modeling language (COSEML) to be used within Component Oriented Software Engineering (COSE) is presented. A Software Engineering paradigm for component oriented system development has been introduced before. The paradigm suggests development by integration rather than code writing. Newly maturing component technology is not fully exploited by the existing Software Engineering approaches. Various representations for object oriented and component related modeling have been investigated and where applicable, established symbols are adopted. Reconciliation of top-down and bottom-up approaches defines the difficulty in component based design. To address top and bottom level concepts, COSEML language acommodates primitives in abstract and implementation levels.

Introduction

Background

The emergence of component technologies [Krieger and Adler 98] opened an exciting era in software development. The notion of building by integration [Tanik and Ertas 97] has been investigated before. JavaBeans offered by Sun Microsystems and ActiveX in Microsoft's DCOM protocol, have been the successful pioneering component technologies. Soon afterwards, component libraries in different domains were available on the market.

Components are units of implemented software building blocks [Szyperski 98]. Despite reported difficulties in their utilization, components are meant for run-time connecting modules. In a purely component oriented approach, assuming that most functions have been implemented in components, the system development problem reduces to locating and connection of defined components. A comprehensive Component Based (CB) methodology is yet to be developed [Brown and Wallnau 98].

Required maturity in the field

Engineering disciplines have convincing success stories with component-based design [Szypersky 98]. An analogy was made to the TTL components in digital circuits field in [Dogru et al. 98]. Maturing of a component-engineering field is regarded as the acceptance of standard naming for components across the industry. Today, a Graphical User Interface (GUI) designer has a good understanding of a 'push button' or a 'combo box.' Also more semantics included in the component interface specifications will aid in the locating and integration of components [Beugnard et al. 99, Della Torre et al. 99].

Contrasting Objects to Components

Objects are abstracted in their classes by a list of properties and methods. Components allocate such lists in their "interfaces," where also "events" are accommodated. In object class hierarchies, variety of methods (and properties) can be grouped in different classes to be inherited. Whereas in Objects, more than one interfaces are allowed to represent one object. An object represents the run-time entity corresponding to a logical model

element, whereas a component corresponds to a structural piece of a bigger software.

For building complex units, objects utilize the *inheritance* concept most of the time whereas components are limited with *composition*. A developer connects a number of components to make a super component. Inheritance is usually a top-down organization while composition is bottom-up. Figure 1 presents the interface perspective for objects and components

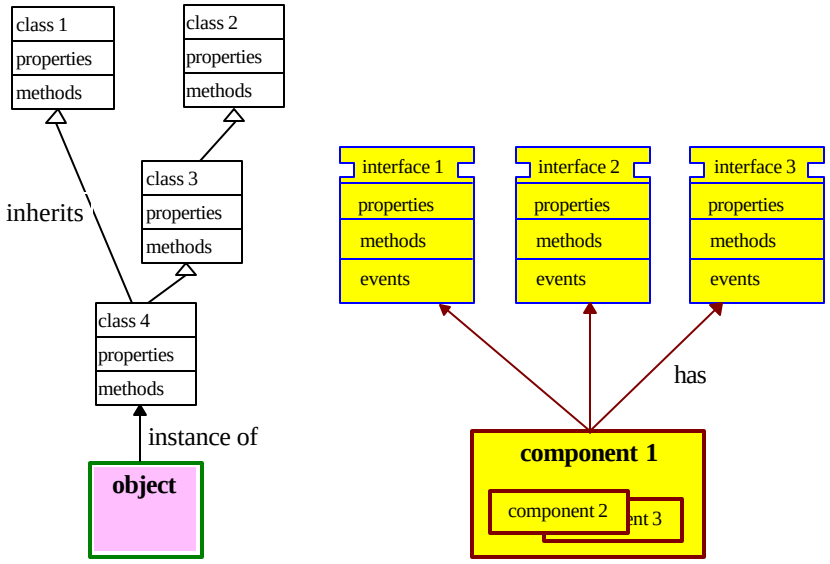


Figure 1: Objects compared to Components

Peculiarities in Component Based integration Approaches

Paradigms need to be supported with Languages and Tools. Methodologies include modeling formalisms and a detailed process prescribing how and when to use the models. The models represent the three dimensions of software systems namely data, function, structure (and sometimes the fourth: control). Traditional approaches have disjoint graphical models for different cross sections of the system based on data, function, and structure. In OO approaches, the mentioned dimensions are represented in combination.

In traditional methodologies a "functional" decomposition of the system is essential. Object oriented methodologies decompose the system with respect to "data". A component-based methodology has to be "structure" oriented more than any other aspect. Figure 2 depicts the modeling emphasis on different dimensions, for different approaches.

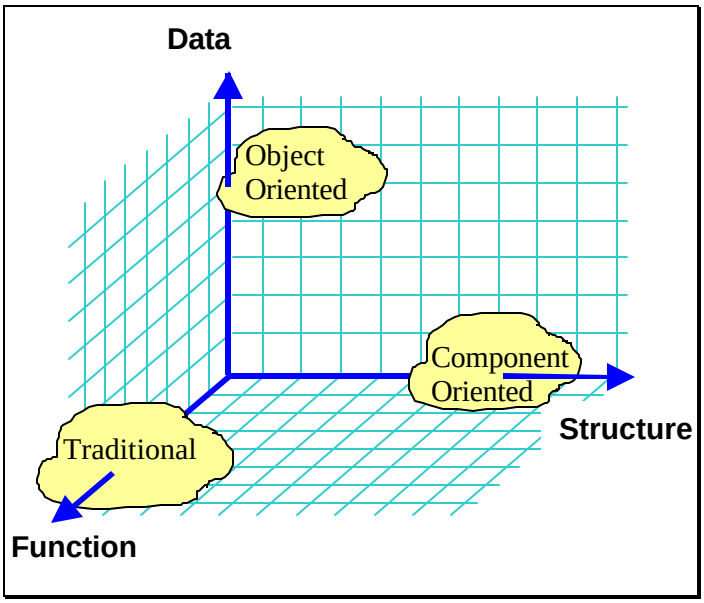


Figure 2. Modeling emphasis for different approaches

COSE Approach

Previous research [Tanik and Chan 91] and established design principles [Simon 69], suggest a top-down approach. Abstract Design Paradigm (ADP) [Tanik and Chan 91] suggested specification environments to decompose the system structure hierarchically (top-down), while accounting for the other dimensions of modeling namely data, function, and control [Dogru et al. 92]. This research capitalizes on the outcome of such former work and foresees the integration of component technologies as a bottom-up patch.

The purely Component Oriented approach starts with a structural decomposition. Meanwhile arriving at existing components is taken into account. Intermediate concepts such as frameworks and design patterns [Gamma et al. 95] can only be included in the process model after obeying a protocol; they can bring in efficiency, with the possibility to distort the pure component orientation. Any such intermediate concept is by default regarded as a super-component in COSE approach. In [Szypersky 98] Frameworks are described as bigger structural elements of a system, than design patterns. However, Design patterns have a higher abstraction level than frameworks. This contradicts the top-down design refinement concept: Bigger chunks should correspond to representations at higher abstraction levels.

Hierarchical decomposition of a system formally, in terms of cubic graphs [Tang et al. 98] has been investigated. The key issue of attaching semantics to the decomposition decisions has been missing. The decomposition should be carried out with respect to two views: abstract design and existing components.

Defining the Language

ADP based work provided data, function, and control kinds of abstractions, superimposed on the structure dimension being the fundamental view. Former key terminology can be adapted to newer mechanisms: The Catalysis approach [D'Souza 98] utilizes Unified Modeling Language (UML) [Booch et al. 99] for a CB representation.

Essentials

COSE transforms the system specification into two entities:

1. A set of components
2. A set of connectors

The set of connectors is a skeleton, connecting the set of components for producing the target system. This network acts as a system-wide 'container' for the components and an interface specification for the entire system. A connector at abstraction level corresponds to a pair of interfaces at component level. A recursive component understanding is preferred, without a container or super-component differentiation.

COSE Modeling Media

COSE Modeling Language "CoseML" is designed and being implemented as the primary modeling input tool. A textual representation is automatically maintained as a background task in the tool begin developed. The grammar for the textual representation is included in the Appendix A.

Model building activity starts top-down, to introduce the building blocks of the system. As the activity continues towards lower granularity blocks, interfaces between the blocks are also defined. At an arrived level where the module is expected to correspond to a component, a temporary bottom-up approach can be taken: If desired capability can only be achieved by a set of components, their integration into a super-component should be carried out.

Graphical modeling elements

Abstract components correspond to ADP "packages," represented by UML's package symbol. Internals of the packages are represented by ADP symbols corresponding to Data, Function, and Control abstractions. A Function abstraction is represented by an oval, similar to UML's Use Case symbol. Implementation level components however require syntax additional to Object representations: *Events* and *Behavioral Descriptions* need to be appended to the *Properties* and *Methods*. Figure 3 depicts basic symbols for CoseML.

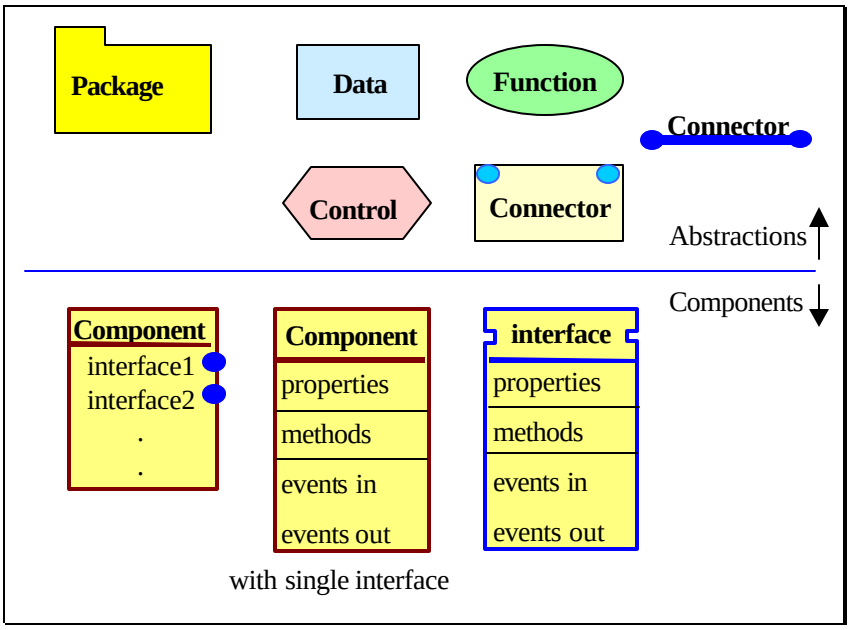


Figure 3. Graphical symbols in the COSE Modeling Language

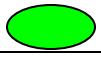
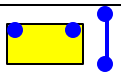
Also needed are the association links to connect the symbols shown in Figure 3. Four kinds of links are used. Among the abstraction elements, the default link is the 'composition,' represented by a diamond at the container end. UML graphical syntax is followed where suitable. Inheritance is represented by a triangular arrow head at the generalized end. Other UML relations can also be represented with additional care. The most special link is a 'connector' represented by a "box" symbol as well as a "line" symbol. The box version is more suitable for interface specification during decomposition. Line version is suitable for information hiding, especially about the interface while visualizing connectivity.

Since the model can contain abstractions and corresponding components, a system function could be represented in different levels. This stereotyping allows for a connector to be represented between two

abstractions, as well as between two components. It is optional to repeat one connector at various levels. A connector between two abstractions, however could correspond to more than one message-links at components level. The lowest-level representation of a message is between specific events or methods of two interfaces. In this case the synchronization semantics for different arrow-head shapes as defined in UML can be utilized.

Compositional links are used among components to construct super-components. Finally, a special kind of association links a component to an abstraction with the 'represents' relation. Table 1 explains further detail about mentioned model elements.

Table 1. COSEML Symbols and their meanings

Symbol	Explanation
	Package is for organizing the part-whole relations. A container that wraps system level entities and functions etc. at a decomposition node. Can contain furhter Package, Data, Function, and Control elements. Also can own one port of one or more Connectors. Can be represented by a Component. The contained elements are within the scope of a package: they do not need connectors for intra-package communication.
	Function represents a System-level function. Can contain further Function, Data, and Package elemenets. Can own Connector ports. Can be represented by a Component
	Data represents a System-level entity. Can contain further Data, Function, and Package Elements. Can own Connector ports. Has its internal operations. Can be represented by a Component.
	Control corresponds to a state machine within a package. Meant for managing the event traffic at the package boundary, to affect the state transitions, as well as triggering other events. Can be represented by a Component.
	Connector represents data and control flows across the system modules. Cannot be contained in one module because two ports will be used by different modules. Ports correspond to interfaces at components level.
	A Component correspond to existing implemented component codes. Contains one or more interfaces. Can contain other Components. Can represent one abstraction (Package, Data, Function, or Control)
	An Interface is the connection point of a component. Services requested from a component have to be invoked through this interface. A port on a connector plugs into an interface
	A *Represents* relation indicates that an abstraction will be implemented by a component An event link is connected between the output event of one interface and the input event of another. The destination end can have arrows corresponding to the synchronization type.
	A Method link is connected between two interfaces to represent a method call. Arrow indicates message direction.
	UML class diagram relations are utilized. Diamond: composition, Triangle: inheritance.

An Example Model

In this section, an example model is presented for a small business automation software. The system functions include:

- Sales,
- Accounting,
- Personnel,
- Clients, and
- Inventory.

Upper level ADP symbols in Figure 4 correspond to logical system components in the specification. Existing implemented components are represented in the lowest level. Textual representation for the model is provided in

the Appendix B.

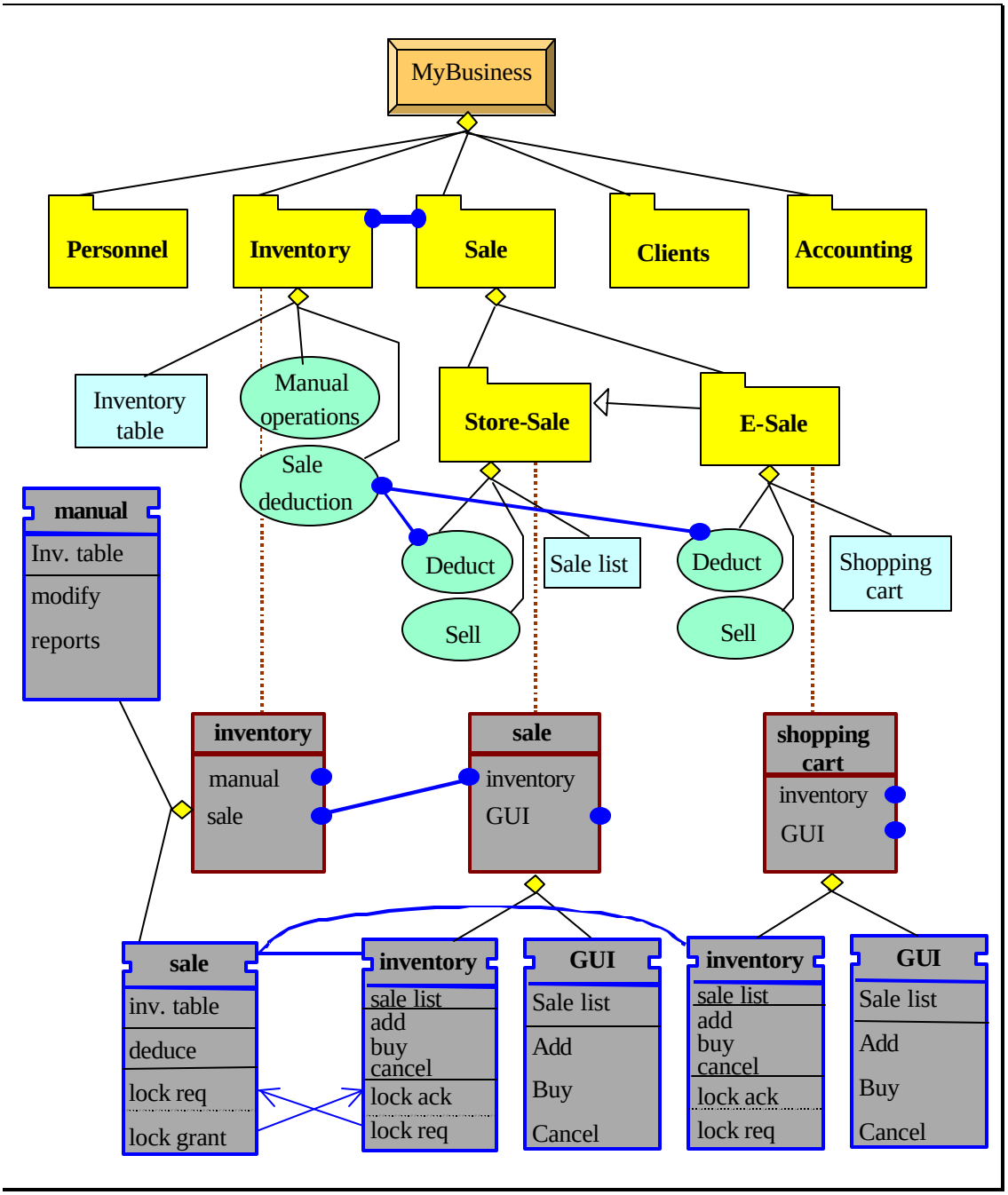


Figure 6. Modeling a small business in COSE

Component oriented modeling will mature with developing rules similar to the modularity principles in design. A foreseen principle is "tangibility." Even when an inheritance relation is used, the inheriting unit is preferred to repeat the inherited assets where suitable. Any unit is desired to be treated individually, as a tangible component.

A Diagram does not have to represent the complete model. Different levels or groups of components could be selected in different diagrams, as well as selecting different kinds of links. The structural break-down will remain as the skeleton, and other views can be turned off and on by selective display of kinds of graphical objects.

Abstraction level packages decompose bigger granularity parts of a system. Function abstractions should be used to represent system functions and capabilities. Data abstractions, as system entities can have their internal operations similar to methods in an object. A unit at higher levels can be represented by a component as its implementation.

Conclusions

Following a concept presentation for a Process Model for the development of software systems based on utilizing available components, a graphical language to support the model is introduced. The language provides primitives to represent logical entities as well as implementation units. An example model is developed to demonstrate the usage of the language.

Future Research

Both the component oriented process model (COSE) and its modeling language COSEML are at their infant era. Case studies are needed to be carried out for justification, and perhaps revision on both the process and the language. A such limited study is already in progress through class projects. Know-how needs to be built to aid in the definition of a complete COSE methodology.

Acknowledgement

The authors express their sincere appreciation for the intellectually stimulating discussions especially by Dr. Tanik, Professors Leon K. Jololian and Osama A.M. Rayis. The implementation help has been invaluable by the classes of CEng 451: Information Systems Development (METU, fall 1999), BIL 343: Object Oriented Programming (Baskent University, 99), and the research, by the graduate class CEng 551: System Design with Abstract Design Paradigm (M.E.T.U, fall 99 and fall 98).

References

[Booch et al. 99] Grady Booch, James Rumbaugh, and Ivar Jacobson, 1999, *The Unified Modeling Language User Guide*, Addison-Wesley.

[Dogru et al. 98] Ali Dogru, Leon Jololian, Franz Kurfess, and Murat Tanik, 1998, "Component-Based Technology for the Engineering of Virtual Enterprises and Software," *Technical Report TR-98-7*, Computer Engineering Department, Middle East Technical University, Ankara Turkey.

[Dogru et al. 92] A.H. Dogru, S. N. Delcambre, C. Bayrak, Y. T. Chen, E. S. Chan, W. Yin, M. G. Christiansen, and M. M. Tanik, 1992, "An Integrated System Design Environment: Concepts and a Status Report," *Journal of Systems Integration* October, Vol, 2, No. 4: pp. 317-347.

[Beugnard et al. 99] Antoine Beugnard, Jean-Marc Jezequel, Noel Plouzeau and Damien Watkins, 1999, "Making Components Contract Aware," *IEEE Computer*, July, Vol. 32, No. 7: pp. 38-45.

[Della Torre et al. 99] Cynthia Della Torre Cicalese, and Shmuel Rotenstreich, "Behavioral Specification of Distributed Software Component Interfaces," *IEEE Computer*, July, Vol. 32, No. 7: pp. 46-53.

[D'Souza, 98] Desmond Francis D'Souza, Alan Cameron Wills, 1998, *Objects, Components, and Frameworks With UML : The Catalysis Approach*, Addison-Wesley.

[Brown and Wallnau, 98] Alan W. Brown and Kurt C. Wallnau, 1998, "The Current State of CBSE," *IEEE Software*, September-.October.

[Gamma et al. 95] Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides, 1995, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Reading, Massachusetts.

[Krieger and Adler 98] David Krieger and Richard M. Adler, 1998, "The Emergence of Distributed Component Platforms," *IEEE Computer*, March.

[Simon 69] Herb A. Simon, 1969, *Sciences of the Artificial*, MIT Press, Cambridge, Massachusetts.

[Szypersky 98] Clemens Szyperski, 1998, *Component Software: Beyond Object-Oriented Programming*, Addison Wesley, New York.

[Tang et al. 98] Y. Tang, A. H. Dogru and M. M. Tanik, "Cyclomatic Complexity Based on Cubic Flowgraphs," *the Third World Conference on Integrated Design and Process Technology, Berlin, Germany*, IDPT Vol. 4, pp. 82-85, July 6-9, 1998.

[Tanik and Chan 91] Murat M. Tanik and Erik S. Chan, 1991, *Fundamentals of Computing for Software Engineers*, Van Nostrand Reinhold, New York.

[Tanik and Ertas 97] M.M. Tanik and A. Ertas, 1997, "Interdisciplinary Design and Process Science: A Discourse on Scientific Method for the Integration Age," *Journal of integrated Design and*

Process Science, September, Vol. 1 No. 1: pp. 76-94.

[Yin 88] Weiping. Yin, 1988, *An Integrated Software Design Paradigm*, PhD. Dissertation, Southern Methodist University, Dallas, Texas.

Appendix A EBNF OF COSEML Version 1.0

Meta-symbol	Meaning
::==	is defined to be
|	alternatively
.	end of production
[X]	0 or 1 instance of X
{X}	0 or more instances of X
(X|Y)	a grouping: either X or Y
"XYZ"	the terminal symbol XYZ

Alphabetic order

```
_____________________________________________________________________
          ascii_string ::== Any_String_of_ASCII_Characters.
              comment ::== ";" ascii_string.
            Component ::== “COMPONENT” id
                           “INTERFACES” id_list
                           ["COMPONENTS" module_id_list].
       condition_exp ::== ascii_string.
            connector ::== "CONNECTOR" id
                           ["PORT-1" id]
                           ["PORT-2" id]
                           ["COMPONENTS" module_id_list].
     connector_desc ::== ascii_string.
     constraint_desc ::== ascii_string.
control_abstraction ::== "CONTROL-ABSTRACTION" id
                           "INITIAL-STATE" id
                           ["STATES" id_list]
                           ["CONTROL-DESC" desc]
                           ["EXCEPTIONS" id_list]
                           ["CONNECTORS" id_list]
                           [“REPRESENTS” module_id_list].
  data_abstraction ::== "DATA-ABSTRACTION" id
                           "FORMAT" format_desc
                           ["CONSTRAINT" desc]
                           ["STORAGE" storage_desc]
                           [data_similarity]
                           ["METHODS" id_list]
                           ["UNIT" desc]
                           ["ACCURACY" desc]
                           ["CONNECTORS" id_list]
                           ["COMPONENTS" module_id_list]
                           [“REPRESENTS” module_id_list].
     data_similarity ::== similarity.
                desc ::== ascii_string.
```

```
                digit ::== 0|1|2|3|4|5|6|7|8|9.
      event_cond_exp ::== "IF" expression "THEN" event_term
                          ["ELSE" event_cond_exp]| event_term.
           event_exp ::== event_exp "OR" event_term
                          |event_term "XOR" event_term
                          |event_term.
               event ::== "EVENT" id
                          ["ORIGIN" id "FEATURE" feature_desc]
                          ["FORMAT" format_desc]
                          ["CONSTRAINT" constraint_desc]
                          ["ACCURACY" accuracy_desc].
         event_term ::== event_term "AND" id|id.
          exception ::== "EXCEPTION" id
                          [except_similarity_spec]
                          [exception_desc]
                          "PRECONDITION" except_condition
                          ["ACTIVE" state_term]
                          ["MESSAGE" event_cond_exp["("id","id")"]].
    except_condition ::== condition_exp |(simple_event_exp | event_term) ["PREEMPT"].
      exception_desc ::== ascii_string.
except_similarity_spec ::== [similarity_desc] except_similarity.
      expression ::== ascii_string.
    feature_desc ::== "PERIODIC" [frequency] | "SPORADIC".
     format_desc ::== ascii_string.
       frequency ::== "CONTINUOUS" [time_spec] | "PULSE" [time_spec].
function_abstraction ::== "FUNCTION-ABSTRACTION" id
                          "INPUT" id_list
                          "OUTPUT" id_list
                          ["PRECONDITION" condition_exp]
                          ["EFFECT" expression]
                          ["SPEC" function_desc]
                          [func_similarity_spec]
                          ["EXCEPTIONS" id_list]
                          ["CONNECTORS" id_list]
                          ["COMPONENTS" module_id_list]
                          [“REPRESENTS” module_id_list].
    function_desc ::== ascii_string.
     func_similarity ::== similarity.
func_similarity_spec ::== [similarity_desc] func_similarity.
               id ::== letter{("_"|letter_or_digit)}.
          id_list ::== id{ id}.
          integer ::== digit{digit}.
        interface ::== “INTERFACE” id
                          [“PROPERTIES” property_list]
                          “METHODS-IN” method_list
                          [“METHODS-OUT” method_list]
                          [“EVENTS-IN” event_list]
                          [“EVENTS-OUT” event_list].
           letter ::== A|...|Z|a|...|z.
  letter_or_digit ::== letter|digit.
           method ::== "METHOD" id
                          "RETURN" data_cond_exp
                          ["OPERAND" id_list]
                          ["PRECONDITION" condition_exp]
                          ["EFFECT" expression]
                          ["OPERATION" op_desc]
                          [op_similarity_spec]
                          ["EXCEPTIONS" id_list].
```

```
                module ::== non_primitive|primitive.
             module_id ::== id.
          module_id_list ::== module_id{ module_id}.
           non_primitive ::== package|data_abstraction|function_abstraction|control_abstraction|connector.
                  number ::== integer|real_number.
                 op_desc ::== ascii_string.
           op_similarity ::== similarity.
      op_similarity_spec ::== [similarity_desc] op_similarity.
                 package ::== "PACKAGE" id
                              "COMPONENTS" module_id_list
                              [package_similarity]
                              [“CONNECTORS” id_list]
                              [“REPRESENTS” module_id_list].
      package_similarity ::== similarity.
               primitive ::== component|interface.
             real_number ::== Any_String_of_Digit_With_a_Single_Embedded_Period.
              similarity ::== "SAME-AS" id | "INSTANCE-OF" id | "INHERITS-FROM" id.
         similarity_desc ::== desc.
       simple_event_exp ::== simple_event_exp "OR" id | id.
                   state ::== "STATE" id
                              ["TRANSACTIONS" id_list].
              state_exp ::== "IF" expression "THEN" id ["ELSE" state_exp | id].
             state_term ::== id | "SELF" | "CONTINUE".
           storage_desc ::== "STREAM"|"BLOCK" integer|"SINGLE"|"SEQUENCE"|"LINKED".
              time_spec ::== ["MAX-TIME"] number.
            transaction ::== "TRANSACTION" id
                              ["EVENTS" event_exp]
                              ["SERVICE" id]
                              "SERVICE-TIME" time_spec
                              ["INITIAL" id_list "INITIAL-TIME" time_spec]
                              ["MESSAGE" event_cond_exp ["("id"," id")"] "MESSAGE-TIME" time_spec]
                              "NEXT-STATE" state_exp.
;----end of COSEML -----
```

_____ notes _____

Comments are transparent, they can appear anywhere.
Blanks are not transparent.

Depth-first Order

```
_____________________________________________________________________________
                desc ::== ascii_string.
        ascii_string ::== Any_String_of_ASCII_Characters.
             comment ::== ";" ascii_string.
      module_id_list ::== module_id{ module_id}.
           module_id ::== id.
                  id ::== letter{("_"|letter_or_digit)}.
              letter ::== A|...|Z|a|...|z.
     letter_or_digit ::== letter|digit.
               digit ::== 0|1|2|3|4|5|6|7|8|9.
             id_list ::== id{ id}.
              number ::== integer|real_number.
             integer ::== digit{digit}.
         real_number ::== Any_String_of_Digit_With_a_Single_Embedded_Period.
          similarity ::== "SAME-AS" id | "INSTANCE-OF" id | "INHERITS-FROM" id.
              module ::== abstract|primitive.
            abstract ::== package|data_abstraction|function_abstraction|control_abstraction|connector.
             package ::== "PACKAGE" id
                          "COMPONENTS" module_id_list
                          [package_similarity]
                          ["CONNECTORS" id_list]
                          [“REPRESENTS” module_id_list].
  package_similarity ::== similarity.
    data_abstraction ::== "DATA-ABSTRACTION" id
                          "FORMAT" format_desc
                          ["CONSTRAINT" constraint_desc]
                          ["STORAGE" storage_desc]
                          [data_similarity]
                          ["METHODS" id_list]
                          ["UNIT" desc]
                          ["ACCURACY" desc]
                          ["CONNECTORS" id_list]
                          ["COMPONENTS" module_id_list]
                          [“REPRESENTS” module_id_list].
         format_desc ::== ascii_string.
     constraint_desc ::== ascii_string.
       storage_desc ::== "STREAM"|"BLOCK" integer|"SINGLE"|"SEQUENCE"|"LINKED".
      data_similarity ::== similarity.
             method ::== "METHOD" id
                          "RETURN" data_cond_exp
                          ["OPERAND" id_list]
                          ["PRECONDITION" condition_exp]
                          ["EFFECT" expression]
                          ["OPERATION" op_desc]
                          [op_similarity_spec]
                          ["EXCEPTIONS" id_list].
            op_desc ::== ascii_string.
      op_similarity ::== similarity.
 op_similarity_spec ::== [similarity_desc] op_similarity.
     similarity_desc ::== desc.
function_abstraction ::== "FUNCTION-ABSTRACTION" id
                          "INPUT" id_list
                          "OUTPUT" id_list
                          ["PRECONDITION" condition_exp]
```

```
                                ["EFFECT" expression]
                                ["SPEC" function_desc]
                                [func_similarity_spec]
                                ["EXCEPTIONS" id_list]
                                ["CONNECTORS" id_list]
                                ["COMPONENTS" module_id_list]
                                [“REPRESENTS” module_id_list].
          condition_exp ::==    ascii_string.
              expression ::==   ascii_string.
          function_desc ::==    ascii_string.
          func_similarity ::==  similarity.
    func_similarity_spec ::==   [similarity_desc] func_similarity.
               exception ::==   "EXCEPTION" id
                                [except_similarity_spec]
                                [exception_desc]
                                "PRECONDITION" except_condition
                                ["ACTIVE" state_term]
                                ["MESSAGE" event_cond_exp["("id","id")"]].
       except_condition ::==    condition_exp |(simple_event_exp | event_term) ["PREEMPT"].
  except_similarity_spec ::==   [similarity_desc] except_similarity.
          exception_desc ::==   desc.
       except_similarity ::==   similarity.
             state_term ::==    id | "SELF" | "CONTINUE".
       event_cond_exp ::==      "IF" expression "THEN" event_term
                                ["ELSE" event_cond_exp]| event_term.
     simple_event_exp ::==      simple_event_exp "OR" id | id.
                event ::==      "EVENT" id
                                ["ORIGIN" id "FEATURE" feature_desc]
                                ["FORMAT" format_desc]
                                ["CONSTRAINT" constraint_desc]
                                ["ACCURACY" accuracy_desc].
        feature_desc ::==       "PERIODIC" [frequency] | "SPORADIC".
           frequency ::==       "CONTINUOUS" [time_spec] | "PULSE" [time_spec].
  control_abstraction ::==      "CONTROL-ABSTRACTION" id
                                "INITIAL-STATE" id
                                ["STATES" id_list]
                                ["CONTROL-DESC" desc]
                                ["EXCEPTIONS" id_list]
                                ["CONNECTORS" id_list]
                                [“REPRESENTS” module_id_list].
               state ::==       "STATE" id
                                ["TRANSACTIONS" id_list].
         transaction ::==       "TRANSACTION" id
                                ["EVENTS" event_exp]
                                ["SERVICE" id]
                                "SERVICE-TIME" time_spec
                                ["INITIAL" id_list "INITIAL-TIME" time_spec]
                                ["MESSAGE" event_cond_exp ["("id"," id")"] "MESSAGE-TIME" time_spec]
                                "NEXT-STATE" state_exp.
          event_exp ::==        event_exp "OR" event_term|event_term "XOR" event_term|event_term.
         event_term ::==        event_term "AND" id|id.
          time_spec ::==        ["MAX-TIME"] number.
          state_exp ::==        "IF" expression "THEN" id ["ELSE" state_exp | id].
          connector ::==        "CONNECTOR" id
                                ["PORT-1" id]
                                ["PORT-2" id]
                                ["COMPONENTS" module_id_list].
          primitive ::==        component |interface.
```

```
Component ::== "COMPONENT" id
               "INTERFACES" id_list
               ["COMPONENTS" module_id_list].
interface ::== "INTERFACE" id
               ["PROPERTIES" property_list]
               "METHODS-IN" id_list
               ["METHODS-OUT" id_list]
               ["EVENTS-IN" id_list]
               ["EVENTS-OUT" id_list].
```

;----end of COSEML -----

_____ notes _____

Comments are transparent, they can appear anywhere.
Blanks are not transparent.

Appendix B COSEML Specification Example

Alphabetic Order

```
PACKAGE     MyBusiness
            COMPONENTS Personnel Inventory Sale Clients Accounting
PACKAGE     Personnel
            COMPONENTS
PACKAGE     Inventory
            CONNECTORS SaleToInventory
            COMPONENTS InventoryTable ManualOperations SaleDeduction
            REPRESENTS inventory
PACKAGE     Sale
            COMPONENTS StoreSale ESale
PACKAGE     Clients
            COMPONENTS
PACKAGE     Acounting
            COMPONENTS
CONNECTOR   SaleToInventory
            PORT-1 Inventory
            PORT-2 Sale
            COMPONENTS StoreSaleToInventory ESaleToInventory
DATA        InventoryTable
            FORMAT Database Table
FUNCTION    ManualOperations
            INPUT InventoryTable
            OUTPUT InventorTable
            SPEC  A local administrator should do the modifications to the values in the table
FUNCTION    SaleDeduction
            INPUT SaleList
            OUTPUT InventoryTable
            SPEC Automatically reduces the 'amount' for items sold
PACKAGE     StoreSale
            COMPONENTS Deduct Sell SaleList
            REPRESENTS sale
PACKAGE     ESale
            COMPONENTS Deduct Sell ShoppingCart
            INHERITS-FROM StoreSale
            REPRESENTS ShoppingCart
FUNCTION    Deduct
            INPUT SaleList
            OUTPUT InventoryTable
            SPEC Sends a message to Inventory for deducing 'amounts'
            CONNECTORS StoreSaleToInventory.
FUNCTION    Sell
            INPUT SaleList
            OUTPUT SaleList
DATA        SaleList
            FORMAT     Table
            METHODS insert delete modify accept cancel
FUNCTION    ESale_Deduct
            INPUT ShoppingCart
            OUTPUT ShoppingCart
            CONNECTORS ESaleToInventory
DATA        ShoppingCart
```

```
                INHERITS-FROM SaleList
COMPONENT       inventory
                INTERFACES manual sale
COMPONENT       sale
                INTERFACES inventory gui
COMPONENT       ShoppingCart
                INTERFACES inventory gui
INTERFACE       manual
                PROPERTIES InventoryTable
                METHODS-IN modify reports
INTERFACE       sale
                PROPERTIES InventoryTable
                METHODS-IN deduce
                EVENTS-IN LockRequest
                EVENTS-OUT LockGrant
INTERFACE       inventory
                PROPERTIES SaleList
                METHODS-IN add buy cancel
                EVENTS-IN LockAck
                EVENTS-OUT LockRequest
INTERFACE       gui
                PROPERTIES SaleList
                METHODS-IN add buy cancel
INTERFACE       ShoppingCart_inventory
                PROPERTIES SaleList
                METHODS-IN add buy cancel
                EVENTS-IN LockAck
                EVENTS-OUT LockRequest
INTERFACE       ShoppingCart_gui
                PROPERTIES SaleList
                METHODS-IN add buy cancel
```

Depth-first order

```
PACKAGE       MyBusiness
              COMPONENTS Personnel Inventory Sale Clients Accounting
PACKAGE       Personnel
              COMPONENTS
PACKAGE       Inventory
              CONNECTORS SaleToInventory
              COMPONENTS InventoryTable ManualOperations SaleDeduction
              REPRESENTS inventory
CONNECTOR     SaleToInventory
              PORT-1 Inventory
              PORT-2 Sale
              COMPONENTS StoreSaleToInventory ESaleToInventory
DATA          InventoryTable
              FORMAT Database Table
FUNCTION      ManualOperations
              INPUT InventoryTable
              OUTPUT InventorTable
              SPEC  A local administrator should do the modifications to the values in the table
FUNCTION      SaleDeduction
              INPUT SaleList
              OUTPUT InventoryTable
              SPEC Automatically reduces the 'amount' for items sold
COMPONENT     inventory
              INTERFACES manual sale
INTERFACE     manual
              PROPERTIES InventoryTable
              METHODS-IN modify reports
INTERFACE     sale
              PROPERTIES InventoryTable
              METHODS-IN deduce
              EVENTS-IN LockRequest
              EVENTS-OUT LockGrant
;---------- end of Inventory Package ----------
PACKAGE       Sale
              COMPONENTS StoreSale ESale
PACKAGE       StoreSale
              COMPONENTS Deduct Sell SaleList
              REPRESENTS sale
FUNCTION      Deduct
              INPUT SaleList
              OUTPUT InventoryTable
              SPEC Sends a message to Inventory for deducing 'amounts'
              CONNECTORS StoreSaleToInventory.
FUNCTION      Sell
              INPUT SaleList
              OUTPUT SaleList
DATA          SaleList
              FORMAT     Table
              METHODS insert delete modify accept cancel
COMPONENT     sale
              INTERFACES inventory gui
INTERFACE     inventory
              PROPERTIES SaleList
```

```
                        METHODS-IN add buy cancel
                        EVENTS-IN LockAck
                        EVENTS-OUT LockRequest
INTERFACE               gui
                        PROPERTIES SaleList
                        METHODS-IN add buy cancel
;----------------  end of StoreSale Package -------------------
PACKAGE                 ESale
                        COMPONENTS Deduct Sell ShoppingCart
                        INHERITS-FROM StoreSale
                        REPRESENTS ShoppingCart
FUNCTION                ESale_Deduct
                        INPUT ShoppingCart
                        OUTPUT ShoppingCart
                        CONNECTORS ESaleToInventory
DATA                    ShoppingCart
                        INHERITS-FROM SaleList
COMPONENT               ShoppingCart
                        INTERFACES inventory gui
INTERFACE               ShoppingCart_inventory
                        PROPERTIES SaleList
                        METHODS-IN add buy cancel
                        EVENTS-IN LockAck
                        EVENTS-OUT LockRequest
INTERFACE               ShoppingCart_gui
                        PROPERTIES SaleList
                        METHODS-IN add buy cancel
;---------------- end of Esale Package --------------------------
;---------- end of Sale Package ----------------------------------
PACKAGE                 Clients
                        COMPONENTS
PACKAGE                 Acounting
                        COMPONENTS
;----end of MyBusiness
```