On Activity-centred Graphs As Linguistic, Structured and Logically Consistent Semantic Networks

Alexandra Galatescu

Research Institute for Informatics
8-10 Averescu Avenue,
71316 Bucharest 1
ROMANIA
e-mail: agal@std.ici.ro

Abstract: The main goal of the paper is to present and advocate for the representation of the activity-centred graphs (ACGs, derived in our research from the classical conceptual graphs) as linguistic, structured and logically consistent semantic networks. We use these graphs within a project on developing a multidatabase CASE environment. Some general benefits of the ACG representation, are underlined: 1) to transcend from pure (complex) object representation to general concepts, meaning not only objects but also properties, events, states, points in time, expressions, structures, abstract knowledge, etc. 2) to unify the data, and the functional and control perspectives in information system modelling and processing 3) to supervise the concept and process interoperability 4) to increase the code and repository reusability. These benefits are mainly due to the ACG ability of representing both the role-like and operator-like dependencies among concepts and of extrapolating such software engineering principles as: abstraction, structuring, modularity, connectivity and encapsulation. The logical consistency and correctness of the ACG representation derives from its natural mapping to predicate calculus and from the natural implementation of the λ-abstraction mechanism.

Keywords: Conceptual Modelling, Conceptual Graphs, Semantic Networks with Node and Link Specialization, Operator-like and Role-like Relationships among Concepts, Knowledge Structuring, Modularization, Connection, Encapsulation

Alexandra Galatescu graduated in Mathematics, from the Bucharest University in 1977. She received M Sc. degree in operations research and mathematical statistics. Since 1977, she has worked at Research Institute for Informatics in Bucharest, where currently she is senior researcher. Her expertise and present research interests are in software technologies, databases and knowledge representation, with emphasis on software tools and technologies for developing complex information systems. She has aquired experience in designing and developing the information system for the Romanian Health Ministry, the National Planning Information System, as well as software tools like: a report generator, two debuggers, a virtual memory device, etc. She initiated at the institute researches in heterogeneous distributed databases. Now she is responsible for a project on developing a multidatabase CASE environment.

1. Introduction

We have defined the *activity-centred graphs* (ACGs) in a project, under way, on developing a multidatabase CASE environment. They are our response to the following conclusions drawn during our research : 1) *natural language is the ideal model* (or metamodel), able to express any information and situation in the real world; 2) the need for transcending from pure (complex) objects to general *concepts*, meaning not only objects but also properties, events, states, points in time, expressions, structures, abstract knowledge or types, etc.; 3) the need for simultaneously representing role-like dependencies and operator-like dependencies as well. *Operator-like dependencies* help us unify three perspectives: data perspective, functional perspective, control perspective (see Section 5). *Role-like dependencies* help us qualify *active concepts* by one or more *attributive concepts* (see Section 2); 4) a *structured and logic-based language* is necessary for mastering the complexity of concept and process interoperability and helping in its automated capturing and interpretation.

The first two conclusions above directed us to the conceptual graph (CG) formalism (see [Sowa 88], [Sowa91], [IRDS95], [Catach85], [Fargues86], [Chein92]), mainly dedicated to natural language processing. The last two conclusions made us 1) transform the classical conceptual graphs (CGs, that represent only role-like dependencies among concepts) into activity-centred graphs (that define only operator-like dependencies); 2) unify the ACGs and the CGs in, what we called, *activity-centred graphs with qualified concepts*; 3) extrapolate the software engineering principles (abstraction, structuring, modularity, connectivity, encapsulation) to the ACG representation.

The paper is organized as follows: Section 2 presents the activity-centred graphs as linguistic semantic networks with node and link specialization; Section 3 analyses several ACG structuring techniques and makes some considerations on ACGs; Section 4 is meant to prove the logical consistency and correctness of the ACG representation, and Section 5 synthetises the main benefits from the ACG usage. Figures are placed at the end.

2. Real World Abstraction Using Linguistic Activity-centred Graphs

Real world abstraction is the first step toward its automation. Any new conceptual model, language or representation formalism, including the proposed activity-centred graphs, means a new abstraction technique. In our acceptance, activity-centred graphs, viewed as linguistic, structured and logically consistent semantic networks, are supposed to represent any (existing or future) type of concepts and dependencies among concepts.

2.1 Node Specialization in the Activity-centred Graph Representation

An activity-centred graph consists of two kinds of nodes:

- an *operation,* that abstracts a declarative or execution activity in a target process/ system, and
- one or more *concepts,* that define or describe the operation. A concept may be *primitive* (represented by a value) or *structured* (with a structured definition, represented by another activity-centred graph). As concepts, we may represent: objects, properties, abstract knowledge or types, events, states, points in time, structures, expressions, etc. A concept is denoted by [CONCEPT_TYPE: referent], where CONCEPT_TYPE is an abstraction (e.g. PERSON) of a real world entity (e.g. 'John'), called 'referent' (or 'individual'). In an activity-centred graph, *referent* may be: 1) an individual concept; 2) a set of concepts put into braces $\{c_1, \ldots, c_n\}$ or 3) a generic concept denoted by its absence.

In the activity-centred graph definition (see Figure 2-1), the concepts are connected to the operation by concept-operation links (see Section 2.2). In Section 2.2.3 the *active concepts* (direct participants in operations, i.e. directly connected to the operation) are separated from the *attributive concepts* (devoted to qualify the active concepts).

2.1.1 Constraints on Concept Nodes

Some of the constraints imposed on concepts in the classical conceptual graph theory have been adapted to the modelling requirements. They are not exhaustive but, so far, they have been sufficient for the multidatabase modelling. The following constraints on the concept nodes have been proposed:

- *concept typing,* which compels each individual concept to comply with a certain (usually domain specific) type. In [IRDS95], higher-order types are proposed for conceptual graphs. A *higher-order type* has as referent another concept type (usually at a lower abstraction level), instead of an individual concept. For example, [PERSON: TEACHER] and [TEACHER:'John Smith']. PERSON is a second order type, whose referent is the first order type TEACHER.
- *concept quantification.* In the classical conceptual graphs (and in predicate calculus), the *universal quantifier* ($\forall$) meaning 'all', 'every', 'any' and the *existential quantifier* ($\exists$) meaning 'there exists' seem to be sufficient for narrowing the spectrum of the individual concepts. An additional specification was needed for the concepts involved in activity-centred graphs, mainly dedicated to concept/ process modelling. It refers the *compulsory or optional existence* (or acquisition) of the referents which instantiate the existentially quantified concept types. Consequently, we refined the existential quantifier to represent: 1) *compulsory existence* of the referent, denoted by $\exists$ and meaning 'must exist' and 2) *optional existence* of the referent, denoted by $\exists?$ and meaning 'may exist'. In the ACG representation, the quantifier precedes the concept type (see Figure 2-1).
- *type of the concept plural.* Each concept may be *singular* (an instance) or *plural* (more instances). From among the four types of plural defined in the conceptual graph formalism ([IRDS95]), we have extracted and adapted the following two plural types:

- *collective plural*, whose elements altogether define the respective plural concept. We denote it by C{}. In case of a finite number of predefined instances composing the plural concept, we may specify them in braces $C\{r_1, \ldots, r_n\}$.

- *distributive plural*, whose elements separately define the plural concept and participate in the graph. We denote it by D{} or $D\{r_1, \ldots, r_n\}$.

- *cardinality of the concept plural*, that limits the number of instances of a plural concept, be it either collective by C@n{} or distributive by D@n{}.

- *concept subtyping*, that endows each concept type with lower-level meanings in the modelled domain, generally not disjunctive and composing a lattice for each concept type. In Section 3, all the proposed kinds of subtyping are defined: 1) by *concept restriction* to particular values, 2) by *concept specialization* to specific roles in a certain domain, 3) by *concept renaming* by synonyms in other languages or ontologies.

2.1.2 Constraints on Operation Nodes

Each operation is liable to the following constraints:

- *operation modality*, denoted by the 'modal' verb that precedes the operation name (see Figure 2-1). In the ACG representation, it specifies a compulsory ('must') or optional ('may') activation of the operation.

- *operation subtyping*, by replacing the activation of an operation (possibly, a virtual one) by the activation of one of its specialized children, connected by the interoperation connective SPEC to a parent operation (see Section 2.2).

- *operation duration*, denoted by a concept that has the role <<DUR>> relative to that operation.

- *operation starting point in time*, denoted by:

  - a concept that has the role <<TIME>> relative to that operation; or by

  - a 'procedural time condition' connected, by the interoperation connective TIME, to the respective operation.

- *operation stimulating event*, denoted by:

  - a concept that has the role <<EVNT>> relative to that operation; or

  - an 'event operation' connected, by the interoperation connective EVNT, to the stimulated operation.

- *operation procedural goal*, denoted by a subsequent operation (connected by the interoperation connective GOAL) that represents the procedural motivation of the operation.

2.2 Link Specialization in the Activity-centred Graph Representation

2.2.1 Concept-Operation Link

This kind of a link connects a concept to the operation it is involved in. In some previous papers, we have called it *conceptual connective*, meaning the *role* of the concept *relative to the respective operation*. Each conceptual connective represents a complementary meaning of a concept and, also, a label to the graphic conceptual link (arrow or line) between the concept and the operation (see Figure 2-1).

As acronyms and meanings of these roles, we propose: AGNT (an agent of the activity, i.e. the person or the object that performs the activity), PTNT (patient, i.e. the object the activity operates on), RSLT (the result of the activity), INST (an instrument for achieving the activity), RCPT (a recipient, i.e. the person or the object benefiting the result of the activity), LOC (location of the activity or of the involved concept), CHRC (a characteristic of the activity or of an involved concept), SRC (the source of an activity or its initial state), DEST (the destination of an activity or its final state), PART (part of another concept meaning a whole), TIME (the point in time when the activity starts), DUR (the duration of the activity), EVNT (an event that stimulates the activity execution, represented as a concept),

DSCR (the activity description, represented as a concept), STAT (the state of the activity), CAUS (the activity cause, represented as a concept), GOAL (the activity goal, represented as a concept). The list may be enlarged, depending on the specific domain which they are used for. These roles have *linguistic synonyms* such as: 'by' for AGNT, 'of', 'upon' for PTNT, 'into' for RSLT, 'with' or 'through' for INST, 'at' for TIME, 'when' for EVNT, 'as', 'of' for CHRC, 'in', 'on' , 'over' for LOC, 'from' for SRC, 'to' for DEST, 'of', 'out of' for PART, 'like' for DSCR, 'for' for RCPT, 'for' for DUR, 'in', 'as' for STAT, 'because', 'because of' for CAUS, 'to', 'in order to' for GOAL.

Figure 2-1 represents the concept-operation links within an activity-centred graph in graphic, linguistic and linear notations.

As an operation description generally takes two or more concepts, a named tuple like OPERATION(CONC_TYPE$_1$ ..... CONC_TYPE$_n$) represents an *operator-like relationship* between concepts and may at once be an *operation signature*. A further abstraction of the operation signature, with an impact on the operation reusability, is the named tuple

OPERATION (role$_1$, ..., role$_n$).

The concepts roles in the operation description are invariant, whereas the concept types generally depend on the specific domain which they are defined for.

2.2.2 Interoperation Link

It connects two or more operation nodes. We called such links *interoperation connectives* (IOC), meaning *constraining or control rules/ statements* on the operation activation. We classified the interoperation connectives according to 1) control statements in the structured programming (sequential, conditional and iterative control), and 2) new constraints required for multidatabase modelling or foreseeable in the multidatabase execution (see also the execution model described in [Galatescu97]). In summary, we propose several interoperation connectives dedicated to:

- ***sequential activation*** of the operations, refined into:
  - *activation timing*, by THEN, BFOR, AFTR between a parent operation and its children, between brothers or between operations belonging to different subtrees in the process decomposition;
  - *explicit specification on compulsory or optional activation* of the child operations, by MUST or MAY;
  - *semantic specialization* of a parent operation by SPEC, which constrains one of the specialized child operations to get activated instead of a parent operation (which might be a virtual operation);
  - *imperative subsequent activation*, by DO or RSLT, specifying compulsory activation of an operation, immediately after the activation of the operation it is connected with, by the respective IOC;
- ***conditional activation*** of the operation by:
  - IF-THEN-ELSE, when the execution of an operation (preceded by THEN or ELSE) depends on a continuation procedural condition (preceded by IF). A *continuation procedural condition* is a predicate expression whose components are operations and logical operators. The true value of this condition depends on the true values of the component operations. An operation is true if already activated and false if not.

NOTE. Using only procedural conditions in the conditional (as well as iterative, see further) control of the operation activation sounds restrictive. A *continuation conceptual condition* (an expression whose components are concepts and operators) is also needed in a completely structured modelling. In [Galatescu97], the continuation conceptual condition is viewed as a decisional element for the operation activation, part of an implicit (not explicitly represented) IF statement. The true value of the conceptual condition helps the modeller (or the modelling-dedicated interpreter) continue the manual or automated activation of the indicated operations.

In order to keep the representation uniform and to separate the procedural conditions from the conceptual ones, *only the procedural conditions will be preceded by IF*, which acts as a conditional statement in our representation. The conceptual conditions represent additional information attached to the existing interoperation connectives.

- CASE, that specifies an *alternative* (but not necessarily disjunctive) *activation* of the child operations. This connective may be associated with a conceptual condition, whose true value orientates the modeller's decision on the appropriate child operation.

- ***iterative activation*** should be:

  - *conditional*, by WHILE-DO, when the execution of an operation (preceded by DO) depends on a continuation procedural condition (preceded by WHILE);

  - *unconditional*, compulsory/ optional, by MUST/MAY REPEAT applied to a repeatable operation;

- ***logical activation***, which depends on logical connections among operations belonging to the same procedural module (see Section 3). Logical activation may be refined into:

  - *(exclusive) disjunctive activation*, by OR (XOR), when an operation activation logically rejects the activation of other operations connected with it by OR or XOR.

  - *activation conjunction*, by AND, when two or more operations may all be activated ;

  - *activation negation*, by NOT, when we have to represent an explicit rejection of an operation.

- ***grouped activation***, which constrains more semantically related operations on being the children of the same operation, hence on belonging to the same semantic group of operations. The common parent operation and its children are correlated by the connective GROUP. The parent operation augments the meaning of each child with its own meaning.

- ***purpose subjunctive activation***, by GOAL. GOAL is preceded by an operation whose activation is motivated by the execution of another operation (called *subjunctive operation of purpose*) that follows GOAL.

- ***operation description***, by DSCR, which introduces the description of an operation, as one or more correlated child operations;

- ***activation stimulation***, by EVNT, which introduces an *event operation* that stimulates another operation;

- ***activation starting***, by TIME, which introduces the starting point in time of an operation as a *time condition* (a continuation procedural condition whose true value is automatically associated with a certain point in time).

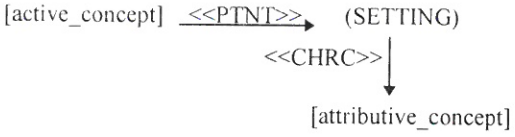
2.2.3 Interconcept Link

Two kinds of interconcept links are represented, meaning:

- ***role of an attributive concept in qualifying an active concept***. This *role-like relationship* between two concepts may simply be represented by an *attributive graph like*

[attributive_concept] —role→ [active_concept]

For procedural purposes, this kind of a link may be simulated by means of a generic (SETTING) operation:



- ***coreference link*** between two concepts referring to the same individual in two separate graphs (see Figure 2.2). Notice that the coreferent concepts might have different types (labels) in the two graphs they belong to.

2.3 Linguistics in the Activity-centred Graph Representation

Activity-centred graphs can represent: *nouns* (as concepts), *verbs* (as operations), *modal verbs* (preceding the operations, see Figure 2-1), *pronouns* (as concepts coreferent to the related nouns, see Figure 2-2), *adjectives* (as attributive concepts introduced by interconcept qualifying roles), *adverbs* (as concepts introduced by the concept-operation link <<DSCR>> or <<CHRC>>), *prepositions and conjunctions* (as

concept-operation links), the *noun plural* (represented by the concept (collective or distributive) plural, see Section 2.1.1).

The ACG linguistics and semantics is upgraded by 1) *concept subtyping* (see Section 3.1), which introduces the *concept polysemy* ([Sowa88]); 2) *concept homonymy* (alternative disjunctive meanings of a concept) by means of, what we call, a multiple concept (see Section 3.1); 3) *anaphoric sentences* (by means of coreferent graphs, see Figure 2-2), 4) *active and passive voice* (by operation and role reversing upon the same storing structure, see Section 5).

2.4 Accommodating the Classical Conceptual Graphs

The definition of the activity-centred graph includes:

- more *elements inherited from the classical conceptual graph representation* and accommodated to our requirements such as:
  - simultaneous representation of two (or more) abstraction levels: the concept type and its individual (possibly another concept type), with benefits in the representation of higher-level concepts;
  - notions of quantification and plural type of the concepts (with a changed position in our notation: we represent them before the concept label in order to keep the user informed. In classical conceptual graphs, they are represented instead of the referent(s) that should replace them).
  - most of the predefined thematic roles (AGNT, PTNT, etc.) that, in classical conceptual graphs, usually express the conceptual relations (role-like relationships directly connecting the concepts to each other);
  - general rules for graph expansion, contraction and joining;
  - graphic notation for concepts;
  - star graph notion (later introduced in [Chein92]);
  - concept subtyping by restricting the concept types to certain values;
  - coreference link between two graphs.
- *modifications on the classical conceptual graphs*, mainly intended to separate the declarative semantics from the procedural one in our representation. These modifications consist of:
  - representation of the *operation* as an operator-like relation between concepts, instead of the role-like relations expressed by thematic roles (PTNT, INST, etc.), prepositions, conjunctions, etc. in the classical CGs.
  - transformation of the *thematic roles* (that compose the starter set of conceptual relations in classical conceptual graphs) *into conceptual connectives*. In an activity-centred graph, they denote the roles of the concepts relative to an operation, instead of the direct roles between concepts, as in Sowa's CGs.
  - explicit representation of the *operation modality* (by 'must' or 'may');
  - definition of the *concept orientation* (input,output, local) in the relation (in our case, operation) signature;
  - separation of generic (domain independent) operations from the domain dependent ones;
  - the transformation of the *intersentential relations and metarelations* among classical CGs *into interoperation connectives*, denoting the procedural control over activity-centred graphs (see Sections 2 and 3).

3. Structuring Techniques and Considerations on Activity-centred Graphs

Abstraction, structuring and encapsulation (information hiding) are major principles of software engineering. Modularity and connectivity (coupling and cohesion) are important aspects of the structuring principle. Section 2 dealt with the abstraction ability of the activity-centred graphs. This Section will analyse the structuring and encapsulation of the activity-centred graphs.

3.1 Concept Structuring and Encapsulation

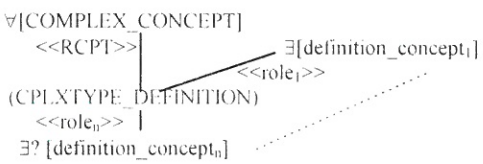
As the data structuring is important for structured programming, similarly, the concept structuring is, in our opinion, a straightforward way to simplifying the conceptual modelling.

3.1.1 Concepts With Structured Definition

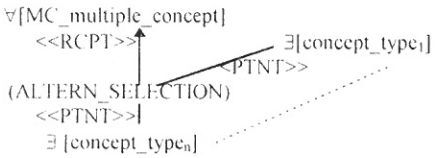
A *concept with structured definition* (or simply a structured concept) is the recipient (and at the same time the result) of a *definition operation* grouping all the concepts (called *definition concepts*) that define the respective structured concept. The abstraction capabilities of the activity-centred graphs allow the natural definition of the structured concepts, independently of the domain specific type of their definition concepts, but depending on the invariant roles of the definition concepts relative to the structured concept. This feature enhances the *reusability of the structured concepts* and makes it possible to share them among more domains/ ontologies/ processes. It is also a step towards a higher-level data structuring, if compared with the allowed structures of today's ordinary programming.

The following kinds of *structured concepts* are proposed (see also [Galatescu95]):

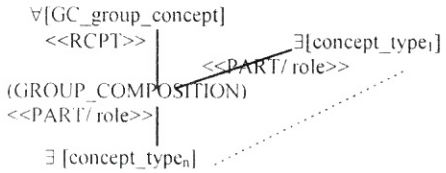
Complex Concept, whose definition associates other concepts (e.g. identification concepts, properties, etc. needed for retrieving the complex concept) related to the respective structured concept in a definition graph like:



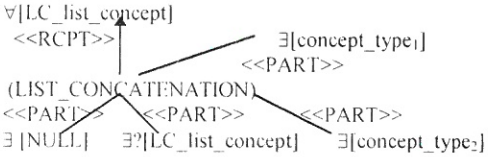
Multiple Concept, that unifies the alternative (disjunctive) meanings of a concept. Only one meaning is selected at a time during the modelling (acquisition) process. The multiple concept implements the *concept homonymy*([Sowa88]). It was introduced in [Fargue86] as a schematic cluster composed of ordered disjunctive schemes. A multiple concept may be defined by an activity-centred graph like:



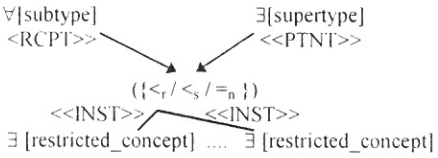
Group Concept, that groups heterogeneous concepts with common higher-level semantics and identical (e.g. PART) or different roles in the group (e.g. concepts that compose: 'person's description', 'object dimensions', 'government', 'family', etc.). A group concept may be defined by a graph like:



List Concept, representing the concatenation of two or more concepts (including a NULL concept, as list terminator). It may be defined by a *recursive operation* represented by a graph like:



Subtype, as a special kind of structured concept defined by a classification operation. On modelling multidatabase, subtypes by restriction and specialization are needed. The *restriction of a concept* means its instantiation with a certain value (in the case of a primitive concept) or the instantiation of one or more definition concepts (in the case of a complex concept) with certain values or subtypes. The *specialization of a concept* means its replacement (in at least one of the graphs containing it) by one of its *specialization roles* in the modelled domain. Synonyms in other languages or ontologies might be subtypes obtained by *renaming the concepts*. The three subtyping operations might be represented by a graph like:



where <r/ <s/=n means subtyping by restriction/ specialization/ renaming. In order to distinguish the subtype external representation, we prefix the name of the subtype by restriction (or by renaming) with '<' (or '='). In order to complete the semantics of the subtyping operation by restriction, the *concepts that should be restricted* (in the complex supertype definition) can be specified.

NOTES

1. Concept subtyping by restriction was introduced in the conceptual graph theory.

2. The definition graph of a subtype by restriction is isomorphic to the definition graph of the restricted concept (supertype), whereas the definition graph of a subtype by specialization or renaming is identical to that of the specialized or renamed concept type. Also, in the case of subtyping by restriction or specialization, the subtype's extension (set of instances) is a subset of the extension of the restricted or specialized supertype.

3. In the case of a complex concept type, the extensions of its subtypes by restriction are disjunctive, whereas in the case of the subtypes by specialization, they may partly overlap.

4. In comparison with the class specialization operation met in the O-O models, that entails a new class definition, *subtyping by restriction/ specialization/ renaming is*, in this implementation, just *a virtual operation*, a semantical restriction (or renaming) of an existing concept type, without physically creating a different one.

3.1.2 Structured Concept Encapsulation. Contraction and Expansion of A Structured Concept

Relying on its λ-definition (see Section 4), the contraction and expansion of a structured concept can be defined as interactive tools for implementing the structured concept encapsulation (see Figure 3-1). *Contraction* of a structured concept means the replacement of its definition graph by its name (type). *Expansion* of a structured concept means the replacement of its name by its definition graph, in any of the graphs containing the respective structured concept.

3.2 Operation Structuring and Encapsulation

3.2.1 Operation Signature

The concepts describing an operation may be input concepts, output concepts, input-output concepts or local concepts. Graphically, the concept direction relative to the operation is represented by the direction of the arrow connecting them. Local concepts are connected by a line.

We mean an *operation signature* a named tuple

OPERATION(CONCEPT_TYPE$_1$,..., CONCEPT_TYPE$_k$)

with 'p' the maximum number of concepts that describes the operation and k ($k \leq p$), the number of input/ output/ input-output concepts. The invariant counterpart of the operation signature is the named tuple

OPERATION(role$_1$,...,role$_k$)

When the concept roles differ from one another, the operation signature may be an unordered list of roles.

3.2.2 Operation Encapsulation. Contraction and Expansion of the Operation

The λ-definition of an operation (see Section 4) is the theoretical support for the contraction and expansion of the domain specific operations, and implicitly, of the graphs that describe these operations. Contraction and expansion help the operation encapsulation (see Figure 3-2). *Operation contraction* means the replacement of the graph that defines the operation by the operation name. *Operation expansion* means the replacement (in the diagram containing it) of the operation name by the graph that describes that operation.

3.3 Joining Two Activity-centred Graphs

Nesting the graphs that define the structured concepts involved in a domain specific operation will be possible provided that the graph that describes that operation joins the definition graphs, which expand the respective structured concepts. Further, the definition graph of a structured concept will join the definition graph of each component structured concept and so on. Hence, join is a technique

necessary for structuring the activity-centred graphs.

3.3.1 Join Definition

The *join* of two activity-centred graphs takes the same steps as the join of the classical CGs:

- draw up a maximal common subgraph (e.g. just a structured concept, in our case);
- attach the remaining parts of the two graphs to the maximal common subgraph.

When one of the two activity-centred graphs is a definition graph, the join operation will be as shown in Figure 3-3.

When both graphs describe domain-specific operations, they may join on a common concept or subgraph, see Figure 3-4.

3.3.2 Joining Appropriateness

We will represent joint graphs when two operations are semantically and chronologically related (for knowledge acquisition or for their execution) and when the following situations are true:

- the common concept type is the same in the two graphs;
- the two operations belong to the same procedural module and have the same parent (see Section 3.4);
- none of the operations is incompatible with a previously activated operation. This condition is a cautionary condition, imposed by the grouped encapsulation of the joint graphs (see Section 3.3.3).

Joining graphs is not incompatible with their independent usage in other modelling or execution contexts. *Internal storing of the joint graphs* depends on the usage (stand-alone or grouped) of the joint operations. The decision is similar to storing joint tables in a cluster or only in physically independent, but logically joint tables in relational databases.

3.3.3 Comparing Join With the Coreference Link Between Two Activity-centred Graphs

If comparing the join of two graphs on a common concept with a possible coreference of the two graphs on two coreferent concepts, the conclusion will be that *joining* graphs *is more restrictive* because:

- the corresponding concepts must have the same type;
- join is only justified provided that the joint operations are encapsulated together and, usually, stored together. The coreferent graphs are usually conceived to be stored independently but, if necessary, they can be encapsulated together.

Graphs (I), (II) and (III) in Figure 3-5 are coreferent because the concept [BOY] is coreferent with the concept [PERSON]. Graphs (I) and (II) join on the concept [HOME]. Graphs (II) and (III) join on the concept [SCHOOL]. Without the contextual correlation between them, graphs (I), (II), (III) may be considered as being independent of one another. Operation (LEARN) may be considered as subjunctive (motivation, goal) with respect to operation (GO).

3.3.4 Encapsulation of Joint Operations

The same as with a stand-alone operation, the *contraction* of two joint domain-specific operations means replacing the corresponding joint graph by the name of the join: $\text{OPERATION}_1 \oplus \text{OPERATION}_2$. The *expansion* of two joint operations means replacing the join name by the corresponding joint graphs. The theoretical support for the encapsulation of two joint graphs is the λ-definition of the respective join (see Section 4).

3.3.5 Encapsulation of Two Coreferent Graphs

When necessary, coreference may be treated as a special (less restrictive) join and encapsulated accordingly. Otherwise, the coreferent graphs may be treated as independent graphs and correlated by other means (e.g. concept transfer between the two operations). In the latter case, the encapsulation of the coreferent graphs means the independent encapsulation of the corresponding operations.

3.4 Modular and Procedural Modelling Using Activity-centred Graphs. Module Encapsulation

In this Section, we focus on two aspects of the conceptual modelling using activity-centred graphs:

- how this representation observes the properties of the modular design, and
- how this representation implements a *hybrid design method*, with a procedural method as skeleton, with a functional decomposition of the procedural modules and with a concept-based flow among operations and among procedural modules.

As shown in Section 2, when using ACGs, two abstraction types prevail: concept types and operations. Each *operation* (including the definition operation of a structured concept type) acts *as a primitive executable and uninterruptible function*. *Module* is a complex abstraction type that combines concepts, operations and their links.

3.4.1 Declarative Modules

We call *declarative module*, any combination of concepts, operations, concept-operation links, concept-concept links. Therefore, declarative modules will be stand-alone, joint or coreferent activity-centred graphs (possibly with qualified concepts) that define or describe structured concepts or domain specific operations.

3.4.2 Declarative Encapsulation

This is achieved by the contraction and expansion of stand-alone, joint and coreferent graphs. Consequently, the contraction and expansion of a declarative module are based on its λ-definition, which is identical to the λ-definition of the corresponding correlated graphs (see Section 4).

3.4.3 Procedural Modules

We call *procedural module*, any combination of operations (contracted activity-centred graphs) and interoperation links (interoperation connectives, see Section 2) that, fully or partly, describe a process.

Functional decomposition of a modelling or execution process, following various criteria, entails a hierarchical structure of the procedural modules that describes the respective process. Practice proves that logical or semantic connections among operations, as well as the execution control needed, are more complex to represent than a simple hierarchy can do. On the other hand, visual and on-line interpretation of the executable or control statements is usually required during a modelling/ execution process. Domain specific operations stand for *executable statements* and the interoperation connectives stand for *control statements*, in any procedural module composed of ACGs (see Figure 3-6).

3.4.4 Procedural Encapsulation

This enables a gradual understanding of the modelling/ execution process. It is the result of a unification, inside a procedural module, of all operations executing or describing the same general task. In order to extend the encapsulation to the procedural modules, we differentiate among three kinds of operations:

- *declarative operations*, expanding onto activity-centred graphs only (e.g. $O_{1\text{-}4}$ in Figure 3-6);
- *procedural operations*, expanding onto procedural modules (operation hierarchies) only (e.g. $O_{2\text{-}1}$ in Figure 3-6);
- *mixed operations*, expanding onto both procedural modules and activity-centred graphs (e.g. $O_{2\text{-}9}$ in Figure 3-6).

This paper is mainly concerned with a modelling process. That is why the dynamic aspects of the operation execution, involving event operations, starting points in time, transformed states or execution actions, will not be addressed. These aspects have been tackled in [Galatescu97].

Contraction and Expansion of A Procedural Module. *Contraction* of a procedural module means replacing the respective module by the name (type) of its root operation. Theoretically, the contraction of a procedural module is based on the procedural λ-definition of its root operation. If the root operation is a mixed operation, it has both a declarative λ-definition and a procedural one (see Section 4).

3.4.5 Accommodating the Properties of Modular Design

ACG representation observes the following properties of the modular design:

- representation may be fully decomposed into declarative and procedural modules;
- each procedural module has a single entry point (its root operation);
- each declarative or procedural module performs a single task. The declarative module performs the task of the corresponding operation and the procedural module performs the task of its root operation.
- each module may be separately executed ;
- declarative and procedural modules are encapsulated;
- declarative and procedural modules share global knowledge;
- each procedural module may have a unique exit point if an *exit operation* is added, as the child of all the leaf operations in the respective module.

3.5 Module Connectivity Using Activity-centred Graphs

Beside modularity, the module connectivity is another aspect of the structuring principle. Connectivity is closely related to the principles of module coupling and cohesion. *Module coupling* aims at minimizing the number and strength of the interconnections between modules. *Module cohesion* aims at strengthening the relationships among the internal elements of a module.

In this Section, we briefly analyse: 1) the procedural connectivity (among procedural modules only); 2) the declarative connectivity (among declarative modules only) and 3) a mixed coupling (among the procedural modules and the declarative ones).

3.5.1 Procedural Connectivity

Role of the Root Operation. Using activity-centred graphs, a low or high degree of the procedural module coupling, that is of the interconnections between the called child module and the caller (parent) module, is reflected in the signature of the root operation belonging to the called module (e.g. $O_{2\text{-}1}$ or $O_{2\text{-}9}$ in Figure 3-6). Regardless its type, procedural or mixed, the root operation of a called module belongs to both the caller and the called module. The concepts present in its signature are supposed to be created in a parent (or an ancestor) module and to be transmitted to all the child operations inside the called module and to all its child modules as well.

Coupling Procedural Modules. Conceptual Dependence. Decreasing the number of concepts present in the signature of a root operation means *to decrease the number of concept transfers* between the two modules (caller and called) and to minimize their coupling.

An additional abstraction of the concepts, materialized in their roles relative to operations, contributes to *loosening the strength of the module coupling*. Instead of coupling the transferred concepts by means of their domain-specific names (types), we may couple them by means of their invariant roles relative to the corresponding operations (i.e. the operations that define them in the parent module and the operations that use them in the child module, including the root operation). This feature of the activity-centred graphs helps the *independent processing of each procedural module* relative to both the specific domain which it is defined for and other procedural modules that use the corresponding concepts.

The *conceptual dependence* (concept transfers) of the procedural modules should be sequential and, generally, top-down directed. Otherwise, redundances or errors can occur in the concept creation and transmission among procedural modules.

Cohesion inside Procedural Modules. High degree cohesion among the operations composing a procedural module is explained by three kinds of possible dependences among these operations:

- *logical dependence*, expressed by the interoperation connectives, as procedural constraints on these

operations. See their logical representation in Section 4.

- *conceptual dependence*, consisting of the concept transfers among the operations of the procedural module;
- *functional dependence*, a consequence of the common goal of all the operations inside a procedural module: to perform the function (task) of the root operation.

3.5.2 Declarative Connectivity

There are three kinds of connections between the declarative modules, which use:

- joining the activity-centred graphs on a common concept or subgraph;
- coreference between the activity-centred graphs involving the same individual concept ;
- concept transfer among the activity-centred graphs;

Coupling the Declarative Modules. Regardless their type, the connections among the declarative modules are intrinsic to the definition of the activity-centred graphs and may be represented in any logical language (including predicate calculus, used in this paper). Thereby, they may be considered as *logically coupled*. On the other hand, the coupled declarative modules contribute to achieving the same function (task): either the task of the domain-specific operation or the definition task of a higher-level structured concept. Hence, these graphs may be considered as *functionally dependent*. Finally, concept transfers among the declarative modules contribute to their *conceptual dependence*.

To minimize these connections means to minimize the concept and operation structuring which is, in our opinion, the clue to the ACG representation and to a modelling/execution process simplification. However, we can minimize the common subgraph in a join, the number of coreferent or transferred concepts. We also need to prevent declarative cycles or redundances.

Cohesion inside the Declarative Modules. The strength of the *functional cohesion* of the concepts defining or describing an operation mainly stems from all the concepts participating in the same operation and playing certain roles in that operation and, indirectly, with respect to each other.

3.5.3 Declarative-Procedural (Mixed) Coupling

This is achieved by the permitted interactive expansion of the declarative and mixed operations inside the procedural modules which they belong to. This possibility makes our representation differ from the rules in current programming, that strictly separates the declarative sections (including the definition of the primitive functions and the definition of the data structures) from the procedural ones.

Using the activity-centred graphs, the coupling between these graphs (as declarative modules) and the hierarchical diagrams of operations (as procedural modules) is directly and interactively achieved. In our opinion, these direct declarative-procedural dependences are appropriate to the modelling/ knowledge acquisition/ execution processes. However, a *disadvantage* presents to the modeller who wants to keep trace of the definition operations of the structured concepts. This disadvantage may be overcome with an appropriate and complete guide to and/ or help in the process execution.

4. Logical Consistency and Correctness of the Activity-centred Graph Representation

We mean by *logical consistency and correctness* of a representation formalism, the consistency and correctness of the abstraction and structuring means which that representation offers with respect to a certain objective. The heuristic nature of the knowledge acquisition process pushes us to think of a *two-step evaluation* of the representation consistency and correctness: 1) an *a priori evaluation* of the correctness and appropriateness of the chosen representation support relative to the intended objective, 2) an *a posteriori evaluation* of the consistency and correctness of the representation result, if compared with the intended objective.

These steps automation should rely on a consistent and correct inference upon the chosen representation means. Experience and software engineering literature have already proved the validity of the structured design and

programming, both enforcing the software engineering principles. That is the reason why the correctness of these principles will be implicitly considered when extrapolated to the ACG representation. This Section mainly deals with the consistency and correctness of the logical inference inside both the declarative and procedural modules representing ACGs and their connections.

4.1 Logical Consistency and Correctness of the Declarative Modules

4.1.1 An Activity-centred Graph Is An n-order Predicate

As shown in Section 2, an activity-centred graph describes an operation (a definition operation or a domain specific one). The concepts that describe the operation may be primitive or structured ones. Each operation

OPERATION(CONCEPT_TYPE$_1$,.... CONCEPT_TYPE$_p$)

(and, implicitly, the graph that describes it) is an n-order predicate if there exists at least one structured concept CONCEPT$_i$, $\forall i \in 1{:}p$, which is an (n-1)-order predicate, because its λ-definition contains (n-2)-order predicates and so on. 'p' denotes the maximum number of concepts which define that operation.

4.1.2 The Role of Quantifiers in the Declarative Inference

The explicit representation of quantifiers is important because of their *role in specifying the inference direction*:

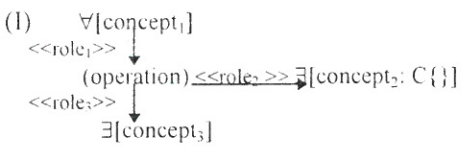
- the universal quantifier preceding a concept indicates the affiliation of the respective concept to the *premise* (left member) *of the inference*. In the case of a plural concept, it means 'for all'. For a singular concept, it means 'any' or 'every'.
- the existential quantifier, meaning 'there exists', obviously indicates an element of the *conclusion of the inference*.

4.1.3 Logical Inference inside Activity-centred Graphs

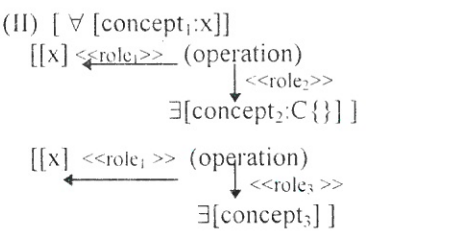
Common sense leads us to the following rules:

- the premise is a conjunction of concepts preceded by a universal quantifier;
- the conclusion is a conjunction of concepts preceded by a (compulsory or optional, see Section 2) existential quantifier;
- the same conclusion should be valid for each component of the plural concept in the premise of the inference;
- an operation uniquely associates a concept (or a component of a plural concept) in the premise with each concept in the conclusion.

These rules suggest such a partition of an activity-centred graph as:



into its premise and the two components of its conclusion, as follows:



After a further transformation of the plural, the inference inside graph (I) becomes:

(III) $(\forall x)\ ((\text{concept}_1(x) \wedge \text{role}_1(x)) \supset (\exists S)\ (\exists z)\ (\text{set}(S) \wedge (\forall y \in S)\ (\text{concept}_2(y) \wedge \text{role}_2(y) \wedge \text{concept}_3(z) \wedge \text{role}_3(z) \wedge \text{operation}\ (x,y,z))))$

NOTES

1. The above rules can be applied to complex declarative modules (joint or coreferent graphs) as well.
2. In formula (III), the conceptual connectives and the plural notations have been transformed into monadic predicates. Variables 'x', 'y' and 'z' have a *double semantics*: that of the concept types they instantiate and that of their roles in 'operation'. This second meaning of the concepts helps represent the graph by a non-positional and domain independent predicate: operation (role$_1$, ..., role$_3$)

3. In the case of an n-order graph, with n>1, the (n-1)-order concept it comprises (e.g. 'z') will be an (n-1) order predicate.

4.1.4 Logic of the Attributive Graphs

An attributive graph like

[attributive_concept] —role→ [active_concept]

that qualifies an active concept will be represented as a dyadic predicate role (attributive_concept, active_concept), as in the case of the classical conceptual graphs. It is a positional, domain dependent and, possibly, an n-order predicate.

The logic inside an attributive graph may be decomposed as follows:

[∀ [attributive_concept:a]] [[a] —role→ ∃[active_concept:c]]

and represented in predicate calculus by:

(∀a) (attributive_concept (a)) ⊃
(∃c) (active_concept (c)) role (a, c)

Hence, each attributive concept must be associated with at least one active concept.

4.2 Logical Consistency of the Procedural Modules

The *two-step evaluation of the procedural consistency and correctness* consists of: 1) evaluating the correctness of the theoretical support dedicated to the logical control inside the procedural module. In our case, that means to evaluate the correctness of the proposed interoperation connectives as well as their correct combination inside a procedural module; 2) evaluating the correctness of the represented process and the degree to which it is free of logical incompatibilities. That means, to prevent and check the modeller's or the user's actions inside any procedural module. In this Section, we are concerned with the first step of the evaluation.

4.2.1 Correctness of the Interoperation Connectives

This is, in our opinion, a consequence of their mapping to predicate calculus. Each interoperation connective may be considered a constraining inference rule, whose *premise* is the last activated operation (denoted by PO) and, implicitly, all the previously activated operations for a given subject and whose *conclusion* is the conjunction (or disjunction) of the operations (denoted by O, O_{i1},..., O_{in}) proposed for the modelling/execution continuation.

In order to represent the mapping rules of the interoperation connectives onto predicate calculus, we also denote 1) by x, y two different output concepts of PO operation transferred to its child operations; 2) by $\supset$ the compulsory activation of an operation implied by a former operation; 3) by $\wedge$ the conjunction of the operations with compulsory activation in the process, by $\vee$ the disjunctive (not exclusive) activation of two or more operations, by == > the transformation into predicate calculus of an interoperation connective and by NULL, a null (ineffective) operation.

In the hereinafter proposed inference, we implicitly suppose a transfer of at least an output concept (denoted by 'x' or 'y') of the operation in the premise to the operations that appear in the conclusion of the inference rule. The concept transfer justifies the logical correlation between those operations, aiming at modelling/ acquiring/ operating on the same concept.

Sequential Activation of the Operation

- PO *THEN / BFOR* O == > (∀x) ((PO (x) ⊃ O (x)) ∧ ¬ (O (x) ⊃ PO (x)))
- PO *AFTR* O == > (∀x) (¬(PO(x) ⊃ O(x)) ∧ (O (x) ⊃ PO (x)))
- PO *MUST* Oi_1, ..., Oi_n == > (∀x) ((PO (x) ⊃ Oi_1 (x)) ∧ ...∧ (PO (x) ⊃ Oi_n (x)))
- PO *MAY* Oi_1, ..., Oi_n == > (∀x) ((PO (x) ⊃ (Oi_1 (x) ∨ NULL)) ∧ ...∧ (PO (x) ⊃ (Oi_n (x) ∨ NULL)))
- PO *SPEC* Oi_1, ..., Oi_n == > (¬PO) ∧ (∀x) (Oi_1(x) XOR ...XOR Oi_n(x)), x a common individual concept for Oi_1, ..., Oi_n
- PO *DO / RSLT* O == > (∀x)(PO (x) ⊃ O (x))

Conditional Activation

IF PO (or procedural condition)
THEN O_1 */ELSE* O_2 ==> (∀x) ((PO (x) ⊃ O_1 (x)) ∧ (∃y) (¬ PO (x) ⊃ O_2 (y))), y an individual concept of O_2

- PO *CASE* $Oi_1, \ldots, Oi_n$ ==> $(\forall x)$ $(PO(x) \supset Oi_1(x) \vee Oi_2(x) \vee \ldots \vee Oi_n(x))$

Iterative Activation

- *WHILE* PO (or procedural condition) *DO* O ==> $(\forall x)$ ($(PO(x) \supset O(x))$ $\wedge (\neg PO(x) \supset NULL)$)
- PO *MUST REPEAT* ==> $(\forall x)$ $(\exists y)$ $(PO(x) \supset PO(y))$. To avoid the infinite loops, we consider NULL instead of 'y' when the repetition ends. We have a declarative end of the repetition, that stops after the PO expansion, upon user's request.
- PO *MAY REPEAT* ==> $(\forall x)$ $(\exists y)$ ($PO(x) \supset (PO(y) \vee NULL)$). We have, in this case, a procedural end of the repetition that stops before the PO expansion.

Logical Activation

- PO *AND* O ==> $(\forall x)$ ($(PO(x) \supset O(x)) \wedge (O(x) \supset PO(x))$)
- PO *OR* O ==> $(\forall x)$ ($(PO(x) \supset (O(x) \vee NULL)) \wedge (O(x) \supset (PO(x) \vee NULL))$)
- PO *XOR* O ==> $(\forall x)$ ($(PO(x) \supset \neg O(x)) \wedge (O(x) \supset \neg PO(x))$)
- *NOT* O ==> $(\forall x)$ $(\neg O(x))$, x an input concept of O .

Grouped Activation

PO GROUP $Oi_1, \ldots, Oi_n$, (with compulsory $Oi_1, \ldots, Oi_k$ and optional $Oi_{k+1}, \ldots, Oi_n$) ==> $(\forall x)$ $(PO(x) \supset (Oi_1(x) \wedge \ldots \wedge Oi_k(x) \wedge (Oi_{k+1}(x) \vee NULL) \wedge \ldots \wedge (Oi_n(x) \vee NULL))$)

Operation Description

PO *DSCR* O ==> $(\forall x)$ $(PO(x) \supset (\exists d)$ DSCR(d)), where DSCR(d) = (λd) λ-graph(O)[d], 'd' is an instance of the PO description and 'λ-graph(O)' is the λ-expansion of the description operation O.

Operation Motivation

PO *GOAL* O ==> $(\forall x)$ $(PO(x) \supset O(x))$

Operation Stimulation

EO *EVNT* O ==> $(\forall x)$ ($(\exists e)$ EVNT $(e) \supset O(x)$), where EVNT(e) = (λe) λ-graph(EO)[e], 'e' is a particular event and 'λ-graph (EO)' is the λ-expansion of the event operation EO.

Operation Starting

O *TIME* time_condition ==> $(\forall x)$ ($(\exists t)$ (time_condition $(t) \supset O(x)$)), 't' is a certain point in time when the time_condition becomes true.

4.2.2 Operation Modality

It introduces the deontic logic on the operation activation. In the present ACG representation, it is reduced to the operation obligation (by 'must' and 'may' verbs preceding the operation type). It may be extended to verbs expressing possibility, intention, wish, belief, etc., relative to the operation execution.

4.3 Logical Support for the Declarative and Procedural Encapsulation

Section 3 presented the declarative and procedural encapsulation, as relying on the λ-definition of the declarative modules (stand-alone, joint or coreferent graphs) and of the procedural ones, to be presented in this Section.

4.3.1 λ-definition of A Structured Concept

Each structured concept has a corresponding λ-expression that defines it. Using the linear notation of a generic definition graph, its λ-expression (or the corresponding λ-graph) can be:

STRUCTURED_CONCEPT =
(λx) (DEFINITION_OPERATION)-
  <<RCPT>> $\forall$[STRUCTURED_CONCEPT:x]
  <<$role_1$>> $\exists$ [$CONCEPT_1$]
. . . . . .
  <<$role_n$>> $\exists$? [$CONCEPT_n$]

and, in predicate calculus, the respective λ-expression will have x as a bound variable:

STRUCTURED_CONCEPT= (λx) $(x_1,\ldots,x_n)$
$(\forall x)$(STRUCTURED_CONCEPT(x) $\wedge$
RCPT(x)) $\supset$ $(\exists x_1) \ldots (\exists x_n)$ ($CONCEPT_1(x_1) \wedge$
$role_1(x_1) \wedge \ldots \wedge (CONCEPT_n(x_n) \vee NULL) \wedge$
$(role_n(x_n) \vee NULL)$)
DEFINITION_OPERATION $(x, x_1,\ldots,x_n)$

NULL helps represent the concept optional existence (indicated by $\exists$? in a graphic or linear notation).

4.3.2 λ-definition of A Stand-alone Operation

Similar to a structured concept, each operation has a corresponding λ-expression that defines it. For example, the operation represented in a linear notation:

(OPERATION) -
<<role$_1$>> ∀ [CONCEPT$_1$]
.....
<<role$_p$>> ∃? [CONCEPT$_p$]

has the λ-definition (in linear notation):

OPERATION =
λ (x_{i1},…,x_{ik})(x_{j1},…,$x_{j(p-k)}$) (OPERATION)-
<<role$_1$>> ∀ [CONCEPT$_1$: x_1]
.....
<<role$_p$>> ∃? [CONCEPT$_p$: x_p]

where $x_{i1},\ldots,x_{ik}$ are the concepts (parameters) that are present in the operation signature and $\{x_{i1},\ldots,x_{ik}\} \subset \{x_1,\ldots,x_p\}$. In a predicate calculus, the respective λ-expression will have $x_{i1},\ldots,x_{ik}$ as bound variables:

$$\text{OPERATION} = \lambda(x_{i1},\ldots,x_{ik})\,(x_{j1},\ldots,x_{j(p-k)})$$
$$(\forall x_1)(\,\text{CONCEPT}_1(x_1) \wedge \text{role}_1(x_1)\,) \supset$$
$$(\exists\, x_2) \ldots (\exists x_p)\,(\text{CONCEPT}_2(x_2) \wedge \text{role}_2(x_2) \wedge \ldots$$
$$\wedge\ (\text{CONCEPT}_p\ (x_p) \vee \text{NULL}) \wedge (\text{role}_p\ (x_p) \vee \text{NULL})\,)\quad \text{OPERATION}\ (x_1,\ldots,x_p)$$

4.3.3 λ-definition of Two Joint Operations

Suppose we join on a common concept [CONCEPT] two graphs with the linear notations:

(OPERATION$_1$)-
<<role$_{1\text{-}1}$>>∀[CONCEPT$_{1\text{-}1}$]
.....
<<role$_i$>> ∃ [CONCEPT]
.....
<<role$_{1\text{-}p}$ >> ∃? [CONCEPT $_{1\text{-}p}$]

(OPERATION$_2$)-
<<role$_{2\text{-}1}$>>∀[CONCEPT$_{2\text{-}1}$]
.....
<<role$_i$>> ∃ [CONCEPT]
.....
<<role $_{2\text{-}s}$>> ∃? [CONCEPT$_{2\text{-}s}$]

The λ-definition of their join will have 'x', representing the common concept, as a bound variable:

$$\text{OPERATION}_1 \oplus \text{OPERATION}_2 =$$
$$\lambda(x)\,(x_1,\ldots,x_p)\,(y_1,\ldots,y_s)$$
$$((\forall x_1)\,(\text{CONCEPT}_1(\,x_1) \wedge \text{role}_1(\,x_1)) \supset$$
$$((\exists\, x_2) \ldots (\exists x) \ldots (\exists x_p)\,(\text{CONCEPT}_{1\text{-}2}(x_2) \wedge$$
$$\text{role}_{1\text{-}2}(x_2) \wedge \ldots \wedge \textit{CONCEPT}(x) \wedge \textit{role}_i(x) \wedge \ldots$$
$$\wedge\ (\text{CONCEPT}_{1\text{-}p}\ (x_p) \vee \text{NULL}) \wedge (\text{role}_{1\text{-}p}\ (x_p) \vee$$
$$\text{NULL})\,)\ \text{OPERATION}_1\ (x_1,\ldots\, x \ldots, x_p)\,))\ \wedge$$

$$((\forall y_1)\,(\text{CONCEPT}_{2\text{-}1}(\,y_1) \wedge \text{role}_{2\text{-}1}(\,y_1)) \supset$$
$$((\exists\, y_2) \ldots (\exists x) \ldots (\exists y_s)\,(\text{CONCEPT}_{2\text{-}2}(y_2) \wedge$$
$$\text{role}_{2\text{-}2}(y_2) \wedge \ldots \wedge \textit{CONCEPT}(x) \wedge \textit{role}_j(x) \wedge \ldots$$
$$\ldots\ \wedge\ (\text{CONCEPT}_{2\text{-}s}(y_{\ s}) \vee \text{NULL}) \wedge (\text{role}_{2\text{-}s}$$
$$(y_s) \vee \text{NULL})\,)\ \text{OPERATION}_2\ (y_1,\ldots\, x \ldots, y_s)\,))$$

where $x_1,\ldots,x_p \neq x$ and $y_1,\ldots,y_s \neq x$.

4.3.4 λ-definition of A Procedural Module

Suppose that a procedural module is a controlled hierarchical diagram composed of OPERATION_1, …, OPERATION_n, with OPERATION_1 the root of the module. And O_{ij}-(OPERATION$_i$, IOC, OPERATION$_j$) are the tuples representing direct connections (by means of interoperation connectives) between two component operations: OPERATION_i and OPERATION_j. Each tuple may be represented in the predicate calculus following the mapping rule for the corresponding IOC, presented in Section 4.2.

λ-definition of the procedural module, with OPERATION_1 as root, may be expressed by the conjunction of O_{ij} tuples (and implicitly of the predicate expressions they represent): $\text{OPERATION}_1 = \lambda(x_{i1}, \ldots, x_{ik})\ (\wedge O_{ij})_{\,i,j}$, with the root operation parameters as bound variables. We call *procedural λ-definition of a root operation*, the λ-definition of the procedural module whose root it is.

5. Conclusions: Benefits from Linguistics, Structure and Logic of the Activity-centred Graphs

A Step Towards a Unifying Representation of the Concepts, Processes and Data/Knowledge Flows.

In [Galatescu97], we advocated the unifying ability of the 'operation', and implicitly of the ACGs describing the operations. The main advantage is that we use *a single representation for three perspectives*: 1) *data perspective*, that aims at modelling the essential information that one needs to represent in a system; 2) *functional perspective*, that aims at modelling the functions and data/knowledge flow between functions and 3) *control perspective*, that aims at modelling the dynamic, time-dependent behaviour of a system. Other solutions (the most important ones will be OMT [Rumbaugh91] and KADS [Schreiber93]) use a distinct representation for each perspective.

In short, the *data perspective* defines, by means of general or domain specific declarative operations, any type of primitive or structured concept, the concept behaviour (operations upon concepts) and the concept interoperability (dependencies among concepts). The *functional perspective* decomposes the process into domain specific dynamic operations. During the process

execution (see [Galatescu97]), these operations are correlated by knowledge flows (input /output concepts) and control flows (continuation conceptual conditions). The *control perspective* views the operation as an uninterruptible unit of execution of the process. Each process is a sequence of dynamic operations, controlled by events, by a procedural strategy (interoperation links) and/or by a (human or automated) decision strategy. The operation may determine the transition of the state (extension) of certain concepts, its own state (active, stopped, waiting, etc.) or the state of the whole process. Finally, the conditions whose true values may also determine the state transition, are defined as list concepts obtained by a concatenation operation, see Section 3.1.

A Natural Framework for Supervising the Concept and Process Interoperability

Concept Interoperability manifests 1) when a structured concept is defined by means of other concepts, correlated by a *definition operation*; 2) when some source concepts influence the destination concepts, correlated by a *dependency operation*; 3) when the execution of a *functional operation* interconnects more concepts, previously defined for the respective process. After its declarative supervision during the modelling step, the concept interoperability becomes effective during the process execution.

Process Interoperability is a consequence of: 1) *interprocess events*, when an *event operation* in a process stimulates an operation executed in another process; 2) *interprocess conceptual dependency*, when the source and destination concepts in a dependency operation are defined in distinct processes; 3) *interprocess conceptual transfers*, when two operations in distinct processes transfer concepts (as parameters) from one another; 4) *interprocess conceptual and logical sharing*, when two processes share concepts/ graphs/ rules/procedures in a common knowledge base. Interprocess transfer and sharing are possible due to the code and repository reusability enabled by activity-centred graphs (see below in this Section).

Data/Knowledge Acquisition, Storing, Interrogation and Interpretation

Acquisition and Interrogation. Activity-centred graphs can be used as *stylized* acquisition and interrogation *sentences*, whose logic and semantics are universal, flexible and extensible because of: 1) their approach to natural language, and 2) their inferential capabilities, close to human reasoning and comprehension. Their declarative and procedural structuring contributes to their gradual knowledge and understanding.

Storing. Natural mapping of the activity-centred graphs onto frames makes it natural to store them in widespread relational databases. We have used ORACLE7 Server to store ACGs in a knowledge base of the multidatabase CASE environment and ORACLE Developer 2000 to acquire and interrogate this base. The distinction between the input, output, input-output and local concepts participating in operations allows an integral or partial storing of ACG (e.g. storing local concepts only).

Interpretation. Beside the interpretation of (inference on) a single graph, as in natural language, we use a complex *thematic and multicriterial interrogation and interpretation* of the graphs, involving one or more target subjects. We use a theme-oriented reasoning for checking the modelling correctness and for comparing the distributed schemas. Both checking and comparing are based on predefined themes (structure, semantics, behaviour, dependencies, etc.) on predefined subjects (schemas, classes, attributes, etc.).

Procedural Guiding and Control of the Process Execution. Control statements (interoperation links) and their mapping rules to predicate calculus help 1) guide (predict) a process execution by a forward inference along the predefined procedural modules composing that process, and 2) control (by a backward inference) the correctness of each operation execution. We have already implemented the procedural guiding and control for a multidatabase modelling process, based on a theme-oriented and event-driven reasoning.

Code and Repository Reusability. The *invariant nature of the concept-operation links* (the roles of the concepts relative to the operation) and, hence, the invariant (and also non-positional) signature of the operation OPERATION ($role_1$,...,$role_n$) (a named and unordered tuple of roles) has an important impact on:

Code Reusability, specifically in:

- *operation processing* (acquisition, storing, interrogation, interpretation, update, etc.), independent of the domain specific concept types, but only dependent on their invariant roles in the operation description;

- *concept transfer* between operations (inside a process or among processes), by correlating the invariant roles of the correspondent parameters, but not their domain specific types;

- *thematic interrogation*, only dependent on the invariant signature of the operations composing the predefined themes.

Repository Reusability granted by:

- *direct and reverse interpretation* of the same storing frame, for an initially stored operation and for its reverse operation as well (similar to active and passive voice in natural language);
- *concept/graph sharing* inside/among processes because of their invariant definitions.

REFERENCES

[**Alonso90**] ALONSO, F., MATE J. and PAZOS, J. **Knowledge Engineering Versus Software Engineering**, DATA&KN. ENG., Vol. 5. No.7, 1990.

[**Catach85**] CATACH, L. and FARGUES, J., **Deduction et opération pour le modele des graphes conceptuels**, IBM Paris Scientific Center, 1985

[**Charette86**] CHARETTE, R.N., **Software Engineering Environments**, Concepts and Technology, INTERTEXT PUBL., 1986.

[**Chein92**] CHEIN, M. and MUGNIER, M., **Conceptual Graphs: Fundamental Notions**, REV. D'INTELLIG. ARTIF., Vol. 6, No. 4,1992.

[**Cooke90**] COOKE, K. E., **Towards A Formalism To Produce A Programmer Assistant CASE Tool,** IEEE TRANS. ON KNOWLEDGE & DATA ENG., Vol.2, No. 3, 1990.

[**Eisenst90**] EISENSTADT, M., DOMINGUE J. et al, **Visual Knowledge Engineering**, IEEE TRANS. ON SOFT. ENG., Vol. 17, No. 10, 1990.

[**Fargue86**] FARGUES J., LANDAU, M.-C., DUGOURD, A. and CATACH, L., **Conceptual Graphs for Semantics and Knowledge Processing**, IBM JOURNAL OF RESEARCH AND DEVELOPMENT, Vol. 30, No.1, January 1986.

[**Galatescu95**] GALATESCU, A., **An Activity-based Ontology Using Conceptual Graphs.** Intl. Symp. KRUSE (Knowledge Retrieval, Use and Storage for Efficiency), Santa Cruz , USA, August 1995.

[**Galatescu96**] GALATESCU, A., **An Approach To Representing Procedural Semantics Over Modelling-dedicated Conceptual Graphs**, Proceedings of 10th Congress on Cybernetics and Systems, Bucharest, Romania, August 1996

[**Galatescu97**] GALATESCU, A., **Toward A Unifying Representation of Concepts, Processes and Knowledge Flows Using Activity-centred Graphs**, 7th Workshop on Knowledge Engineering: Methods & Languages (KEML97), Milton Keynes, UK, January 1997.

[**IRDS95**] **IRDS Conceptual Schema, Part 1: Conceptual Schema for IRDS. Part 2: Modeling Language Analysis**, ANSI Report X3H4/93-196, published in 1995

[**Mills80**] MILLS, H.D., **Principles of Software Engineering**, IBM SYSTEM JOURNAL, Vol. 19, No. 4, 1980,

[**Schmidt89**] J. W. Schmidt and C. Thanos (Eds.) **Foundations of Knowledge Base Management**, Contributions from Logic, Databases and Artificial Intelligence Applications, 1989.

[**Schreiber93**] SCHREIBER, G., WIELINGA, B. , and BREUKER J., **A Principled Approach to Knowledge-based System Development**, ACADEMIC PRESS, London, 1993.

[**Shaw90**] SHAW, M., **Toward Higher-level Abstractions for Software Systems,** DATA &KNOWLEDGE ENGINEERING, Vol. 5. No. 7, 1990.

[**Sowa88**] SOWA, J.F., **Lexical Structures and Conceptual Structures**, in J. Pustejovsky (Ed.) Semantics in the Lexicon, KLUWER ACADEMIC PUBLISHERS, 1988.

[**Sowa91**] Sowa, J.F., **Toward the Expressive Power of the Natural Language. Principles of Semantic Networks. Explorations in the Knowledge Representation**, Morgan Kaufmann Publ., 1991.

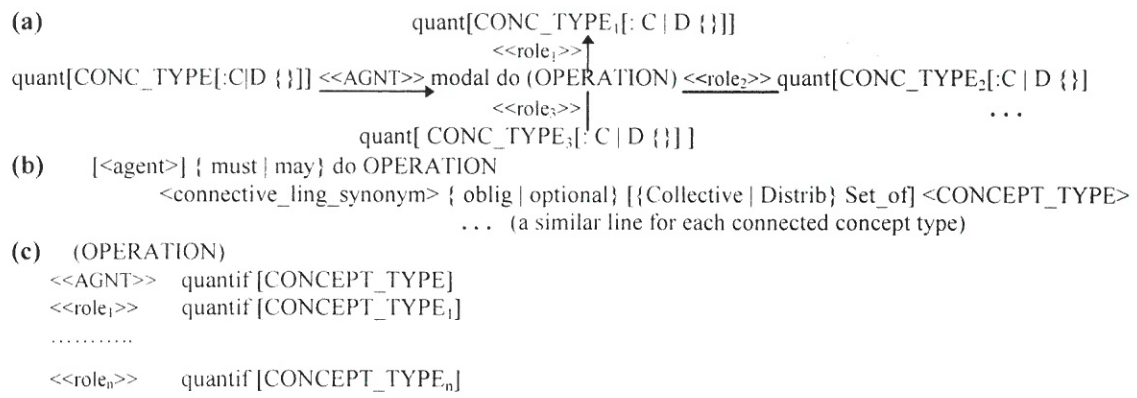


Figure 2-1. An Activity-centred Graph in the Graphic (a), Linguistic (b) and Linear (c) Notations

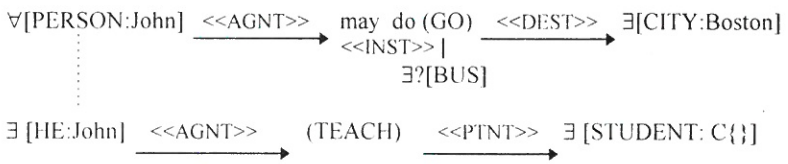


Figure 2-2. Coreference Link between Two Activity-centred Graphs Representing the Situation: "John goes to Boston by bus. He teaches students."

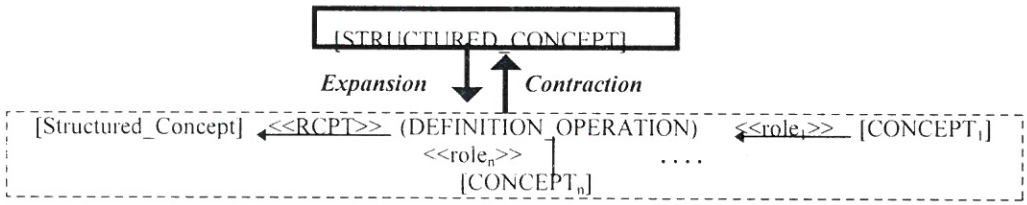


Figure 3-1. Expansion and Contraction of A Structured Concept

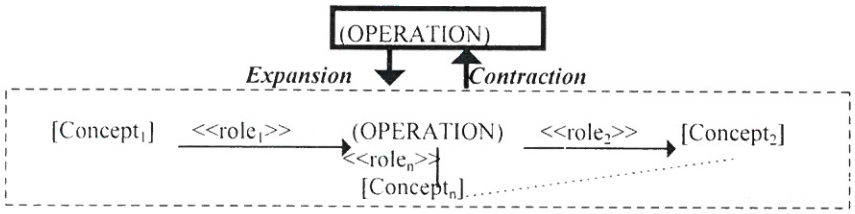


Figure 3-2. Expansion and Contraction of A Domain-specific Operation

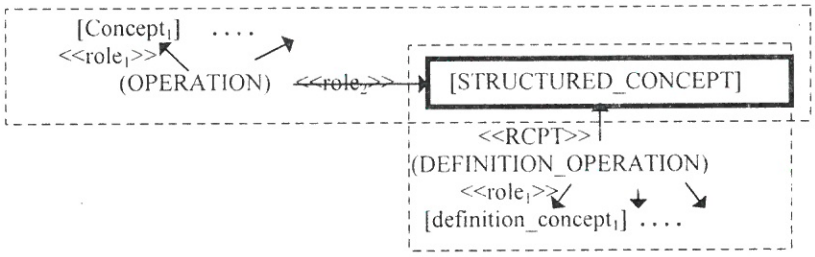


Figure 3-3. Join of A Domain- specific Graph and A Definition Graph

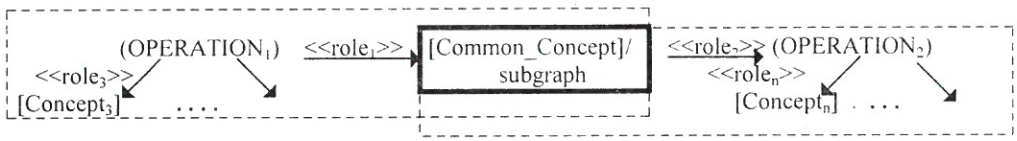


Figure 3-4. Join of Two Domain-specific Graphs

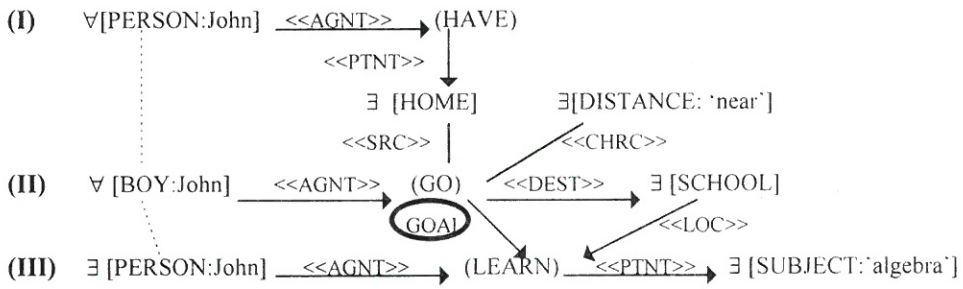


Figure 3-5. Join and Coreference Links (dashed lines) Representing the Situation:

" The boy John goes to school to learn algebra. School is near his home."

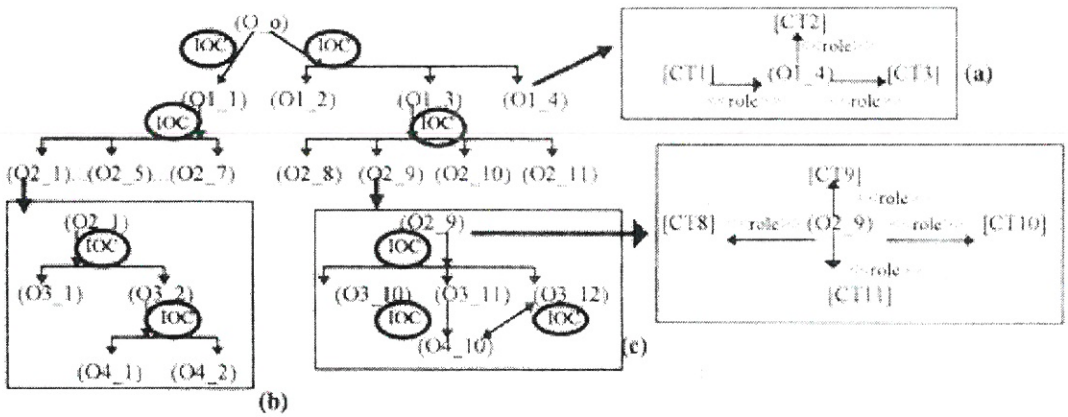


Figure 3-6. Declarative (a), Procedural (b) and Mixed (c) Operation Expansion

O_i - execution statement, IOC - control statement