Databases of Activities and Modeling

Eugen Ardeleanu*
Research Institute for Symbolic Computation
Johannes Kepler University Linz, A-4040, Austria

Adriana Ardeleanu
FAW Research Institute for
Applied Knowledge Engineering

Abstract

The concepts of activity and activity graphs are introduced. The notion of activity is a versatile tool for modeling in general. We will focus on applications to database theory and model-driven programming. The problem of assembling complex components from simple ones is formulated as an equation in the combinatory algebra of (generalized) functionality types. The consistency restrictions are represented by connectivity type matching.

Classification: Model-Based and Qualitative Reasoning, Planning and Scheduling, Intelligent Databases.

1 Introduction

The name *activity* has already appeared many times in artificial intelligence in connection with *planning* and *scheduling*, cf. [2], [11], as well as in operational research in connection with *evaluation* and *optimization techniques*, cf. [1]. In all the contexts in which the denotation "activity" is used, the common interpretation is to designate a basic entity with which, by some sort of assembly process, other activities can be built up in order to *model* and/or *represent* a certain *system*.

In our model, based on an augmented variant of the model described in [5] and [9], an activity has input/output connections and states, and is to be regarded as a cognitive tool to recognize the structure of a data (entity) in a database. The states provide for an internal characterization of the functionality-type of an activity, as in basic algebraic system theory, cf. [7], and are to be regarded as elements of a combinatory algebra which is the mathematical

---

*Supported by the Austrian Ministry for Science and Research.

environment suited to treat local (by connectivity-type parameters) as well as global aspects of the data (by functionality-type). The coupling of two activities is based on input/ output type matching and this is used to direct the search procedure. The concept of activity as defined in this article is a dynamical one supporting model-driven programming.

In Section 2 we present an example of application on a problem originating in robotics and presently dealt with in model-driven programming. In Section 3 we give the generalization and the formal definitions for the concept of activity and activity graphs, point out the differences with other similar concepts present in the literature and show how it supports modelling. Section 4 concludes the article.

2 An Example

The following problem is taken from [6]:

- Write a computer program for a robot to automatically assemble a complex piece, e.g a steering column, out of simple (primitive) components (pieces).

Formulated in model-driven programming terminology this comes down to:

- Compile an exploded diagram from a database of part descriptions and simple assembly tasks into the code needed to drive the robot.

We will now try to point out the relevant aspects of such an enterprise.

1. Each piece can be assembled with several other pieces at fixed connection points. At each connection point only one type of connection is possible (e.g. screwing, insertion, lining) and the connection types of the two pieces assembled at a certain connection point should match.
2. Each piece can serve to several functionalities (e.g the links of a robot can have a rotatorial, translatorial or combined movement, cf. [3]) and the set of functionality types associated to an assembly is an element of a cumulative hierarchy of functionality types.
3. Situations may occur in which the order in which the assembly process is performed is of relevance, i.e assembling in certain (wrong) order may prevent the completion of the process.

3 The Model

In this section we will generalize the considerations from the previous example, give the formal definitions and come up with the model. Finally we will show how this model offers a reasonable solution to the problem mentioned above.

First let us recall some basic notions from [4] and [5]. Starting with a vocabulary of basic facts, let this be FT - the set of functionality types, a cumulative hierarchy of sets of expressions $S_n(FT)$ is built by

$$S_0 = FT$$

$$S_{n+1}(FT) = S_n(FT) \cup \{\sigma \rightarrow s : \sigma \subseteq S_n(FT),\ \sigma\ finite,\ s \in S_n(FT)\}$$

The language for representing the knowledge (about the functionality types as states of activities) is then

$$LS = \bigcup_n S_n(FT)$$

and the objects which we choose as *states representatives* are the subsets of LS. The application of one object (in our interpretation the set of states of an activity) S_A to another object S_B is defined as

$$S_A \cdot S_B = \{s : \exists \sigma \subseteq S_B . \sigma \rightarrow s \in S_A\}$$

The set S_{FT} of all subsets of LS with the binary operation $\cdot$ as defined above, becomes an algebraic structure, a so called *combinatory algebra* $\mathbf{S}_{FT} = \langle S_{FT}, \cdot \rangle$.

In our model the set of states S_A of an arbitrary activity A is simply an element of $\mathbf{S}_{FT}$ and the composition of two activities is modeled by the binary operation $\cdot$.

Consider now a set CT whose elements are called *connectivity types* and for each connectivity type $c \in CT$ a *set of parameters* $P_c = \{p_1, ..., p_{n_c}\}$. Consider also a set S whose elements are called *states*. Identifying $S = S_{FT}$ we are now in the position to give the formal definition for the concept of activity.

An *activity* A consists of a couple (C_A, S_A) where C_A is the *set of connections* and $S_A \in S_{FT}$ is the *set of states* (or the *generalized functionality type*) of the activity A.

The aspects mentioned in Section 2 are formalized as follows:

1. Let A be an activity. Then to each connection $a_i \in C_A$ there is associated a connectivity type $ct(a_i) \in CT$ and a *set of values* $V_{P_{ct(a_i)}} = \{v^1, ..., v^{n_{ct(a_i)}}\}$ for parameters $P_{ct(a_i)}$ uniquely determining the particular connection within the set of connections having the same connectivity type.

2. Activities are the model for the components (pieces) in the example in Section 2. The states of an activity as elements of the combinatory algebra $\mathbf{S}_{FT}$ can be viewed as generalized functionality types and model the functionality type of the corresponding component.

3. The binary operation $\cdot$ in $\mathbf{S}_{FT}$ is non-associative, corresponding to the fact that the order in which the components are assembled is relevant.

The assembly process on components is modelled by the composition of activities via the operation in $\mathbf{S}_{FT}$ under the connectivity type-matching constraint.

Let $A = (C_A, S_A)$ and $B = (C_B, S_B)$ be two activities, $C_A = \{a_1, ..., a_{n_A}\}$, $C_B = \{b_1, ..., b_{n_B}\}$, $n_A, n_B \in \mathbf{N}$ the number of connections of activity A and B, $1 \leq i_1 \leq ... \leq i_m \leq n_A$, $1 \leq j_1 \leq ... \leq j_m \leq n_B$ and $ct(a_{i_k}), ct(b_{j_k}) \in CT$ for all $k = 1, ..., m$, the connectivity types. Then the necessary conditions for the composition of activities A and B at the connections $\{a_{i_1}, ..., a_{i_m}\}$ and $\{b_{j_1}, ..., b_{j_m}\}$ are:

$$ct(a_{i_k}) = ct(b_{j_k})$$

$$v^l_{a_{i_k}} = v^l_{b_{j_k}}$$

$$n_{ct(a_{i_k})} = n_{ct(b_{j_k})} = n_k$$

for all $l = 1, ..., n_k$ and $k = 1, ..., m$.

We assume that at each connection of any activity at most one activity can be connected. In this way the restrictions are put into evidence by considering the connectivity types; a particular connectivity type at a particular connectivity point usually renders inconsistent some functionality types.

The *composed activity* $(AB) = (C_{AB}, S_{AB})$ is defined by:

$$C_{AB} = C_A \cup C_B - \{a_{i_1}, ..., a_{i_m}\} - \{b_{j_1}, ..., b_{j_m}\}$$

$$S_{AB} = S_A \cdot S_B$$

For example, let $\{r(otation), t(ranslation), s(crew)\}$ be a set of functionality types. Then an activity with functionality type $\{r, t\} \rightarrow s$ has the potential that, assembled with an activity with functionality type r or t results in an activity with functionality type s. Further, such one can be obtained by composing an activity with functionality type $\{s \rightarrow (\{r, t\} \rightarrow s)\}$ with an activity having functionality type $\{s, ...\}$, etc. A set of interconnected activities (according to the rules described above) forms an *activity graph*.

Activity graphs are our models for complex components built up from simple ones. The construction of an activity graph can now be represented by an expression in $\mathbf{S}_{FT}$, e.g the expression $A((BC)D)(EF)$, involving the activities A, B, C, D, E, and F, stands for the composition process by which first B is connected with C, then the resulting activity with D, then A with the previous resulted activity, etc. The convention of associativity is on the left.

In this way the result of building an activity graph G from activities $A_1, ..., A_n$ can also be viewed as an activity $G = (C_G, S_G)$ The problem stated at the beginning of Section 2 can now be formulated as:

- Given a (generalized) connectivity type C_0 and a (generalized) functionality type F_0, construct an activity graph G from a database of activities involving activities $A_1, ..., A_n$ such that the equation

$$F_0 = (...((S_G \cdot S_{A_1}) \cdot S_{A_2} \cdots S_{A_n}) \qquad (1)$$

holds in the algebra $\mathbf{S}_{FT}$ and the connectivity type constraints for activities composition are fulfilled.

We note here that the notions "parts descriptions" and "assembly tasks" in the original formulation of the problem were substituted by a single concept of activity and the compilation requirement by the construction of an activity graph, which can be, in principle, effectively computed.

The question now naturally arises, which composition processes can be represented by activity graphs, i.e under what conditions is Eq. (1) solvable for S_G. The answer is known from combinatory algebra (cf. [4]): it is the case if and only if the functional dependence:

$$F_0 = f(S_{A_1}, ..., S_{A_n})$$

is continuous in the following sense:

$$f(S_{A_1}, ..., S_{A_n}) = \bigcup_{\sigma_i \subseteq S_{A_i}} f(\sigma_1, ..., \sigma_n), \quad \sigma_i \ finite$$

and then f can be represented by S_G as:

$$S_G = \{(\sigma_1 \rightarrow (\sigma_2 \rightarrow ...(\sigma_n \rightarrow s)...) : a \in f(\sigma_1, ..., \sigma_n), \ \sigma_i \subseteq S_{A_i}, \ \sigma_i \ finite\}$$

.

The main advantage of this approach is that the problem stated in Section 2., which appears frequently in database theory, can now be stated as an equation in the corresponding combinatory algebra of functionality types with restrictions imposed by connectivity type matching. It is the language of constraint logic programming that successfully deals with such problems, in which the result of each query has to be checked against consistency constraints.

Comparing the concepts of activity and activity graphs as defined in this article with similar concepts in the literature we note that our approach supports type-matching by means of connectivity types (cf. [2]), quantitative analysis and optimization by means of parameter sets (cf. [1]), and equation solving driven inference mechanisms by means of (generalized) functionality types (cf. [4] , [5]). These concepts can be used as classification and comparison techniques (cf. [8]) as well as dynamic problem solving tools.

4 Conclusions and Further Research

We have introduced the concept of activity and activity graphs for modelling complex assembly processes. The construction of an activity graph as a model for a given process is driven by solving an equation in the attached combinatory algebra of generalized functionality types and subject to connectivity type matching constraints.

Even if the construction of an activity graph is effectively computable, the actual implementation seems to be a difficult task. We believe that Prolog, or a Prolog-like programming language with its powerful backtracking and unification/resolution algorithms, could be suited for an implementation.

Another direction for further research could be the investigation of applications in parallelism (computation, manipulation).

Finally we think of applying optimization techniques in the process of constructing the activity graphs.

References

[1] Ackoff, L., Sasieni, W. (1970). *Fundamentals of Operational Research*, Addison-Wesley.

[2] Cohen, P. R., Feigenbaum, E. A., eds., (1982). *The Handbook of Artificial Intelligence*, vol. **3**, Pitman.

[3] Craig, J. J., (1986). *Introduction to Robotics*, Addison-Wesley.

[4] Engeler, E., (1986). Cumulative Logic Programs and Modelling, *Logic Colloquium '86*, Drake, F. R., Truss, J. K., eds., Elsevier.

[5] Engeler, E., (1992). Sketch of a New Discipline of Modelling, *J. Symbolic Computation*, (to appear).

[6] Hopcroft, J. E., (1986). The Impact of Robotics on Computer Science, *Communications of the ACM*, **29**.

[7] Kalman, R. E., Arbib, M., Falb, P., (1965). *Topics in Mathematical System Theory*, McGraw-Hill.

[8] Lockermann, P. C., Moerkotte, G., (1991). On the Notion of Concept, *10-th Int. Conf. Entity-Relationship Approach*, ed., Teory, T. J.

[9] Schwärzler, G. (1992). An Algebraic Approach to Knowledge-Based Modeling, *Conference on Artificial Intelligence and Symbolic Mathematical Computation*, Calmet, J., ed., Springer, LNCS.

[10] Tjoa, A. M., Wagner, R., eds., (1990) *DEXA '90*, Springer.

[11] Vere, S. A., (1983). Planning in Time: Windows and Durations for Activities and Goals, *IEEE Trans. Pattern Anal. Mach. Intell.*, **PAMI-5**.