NOVI SAD J. MATH.
VOL. 30, NO. 3, 2000, 149-160

OBJECT-ORIENTED SPECIFICATION OF THE SYSTEM FOR GENERATING SYMBOLIC MODELS OF ROBOTIC MECHANISMS

Dušan Surla, Miloš Racković[1]
Institute of Mathematics, Faculty of Science, University of Novi Sad
Trg Dositeja Obradovića 4, 21000 Novi Sad, Yugoslavia

Abstract

Automatic generating of the symbolic mathematical models of robotic mechanisms belongs to the class of problems of combinational optimisation. To cope with the mathematical complexity in solving this problem one requires developing of special algorithms and data structures. One way for dealing with this problem is the application of object-oriented approach to design the system for mathematical modelling of robotic mechanisms. This paper presents a formal specification of the system for robotic mechanisms modelling using the Unified Modelling Language.

AMS Mathematics Subject Classification (1991): 68Q40, 70B15
Key words and phrases: object-oriented specification, robotic mechanism models

[1]The work is supported by the Ministry for Science and Techonolgy of Serbia, through the Institute of Mathematics, Novi Sad

1. Introduction

Considerable progress in modelling of robotic mechanisms, compared to numerical methods [1, 2], has been achieved by introducing the numeric-symbolic [3] and symbolic methods which develop special data structures for representing analytic expressions of the model and enable reduction of numerical complexity of the generated model. In [4-7] a database management system was introduced into the process of robotic mechanisms modelling. Navigation through the database and its updating enables calculation of the desired robotic quantities and reduction of the numerical complexity of the model.

The basic Newton-Euler method for forming the model of simple kinematic chain dynamics in closed form [3] has been broadened in such a way as to enable modelling of both complex and closed kinematic chains using the notations introduced in [2]. Now the model of the robotic mechanism in closed form can be expressed as:

$$P = H(q,\Theta)\ddot{q} + \dot{q}^{\top}C(q,\Theta)\dot{q} + + h^{G}(q,\Theta) + B(q,\Theta)\sigma \tag{1}$$

where:

$H = H(q,\Theta) : R^n \times R^m \to R^{n\times n}$ - inertial matrix of the mechanism;

$C = C(q,\Theta) : R^n \times R^m \to R^{n\times n\times n}$ - matrix of the Coriolis and centrifugal effects;

$h^G = h^G(q,\Theta) : R^n \times R^m \to R^n$ - gravity vector;

$B = B(q,\Theta) : R^n \times R^m \to R^{n\times 6}$ - matrix of chain closure;

$\sigma \in R^6$ - reaction vector of chain closure.

A detailed mathematical analysis shows that all model expressions are of the following form:

$$Y = \sum_{i=1}^{N} k_i \cdot \prod_{j=1}^{L} x_j^{e_{ij}} \tag{2}$$

where:

Y - robotic quantity to be calculated;

k_i - constant coefficient related to the i-th addend;

x_j - one of the variables of the robotic mechanism model represented by its name.

e_{ij} - exponent of the j-th variable of the i-th addend.

It is important to mention that the same form (Eq. 2) represents both the basic mathematical expressions for forming the dynamic mechanism model and the fully-developed analytic expressions of the model. The difference is in the fact that x_j in the case of the developed expressions must be one of the basic robotic quantities (q, $\dot{q}$, $\ddot{q}$, $\sin q$, $\cos q$).

This paper uses the object-oriented approach to design a system for symbolic modelling of robotic mechanisms. The formal specification of the system is given in Unified Modelling Language (UML) [8]. The main task of this paper is to develop a data structure for storing all mathematical expressions which represent the model of the mechanism. In other words, we describe the statics of the mentioned system using the object-oriented approach in a class diagram. The state diagram describes the dynamics of the system.

2. Class diagram

The system statics is represented by the class diagram in Fig. 1. This diagram represents the classes which store information about the model expressions.

The basic classes of the system are **Calculating graph** and **Analytic expressions**. The class **Calculating graph** represents mathematical expressions which form the mathematical model of the mechanism dynamics, and the class **Analytic expressions** represents the fully-developed analytic expressions of the model. Both classes have the attribute *phase* which contains the integer value indicating the phase in the process of modelling.

In order to represent the polynomial expressions (Eq. 2) we introduced the abstract classes **Variable** and **Addends**. The class **Variable** is used for representing all variables in polynomial expressions which stand for some really robotic quantities. Each variable has the attributes *code* and *value*. The attribute *code* determines uniquely the variable and the attribute *value* stores the numerical value of the robotic quantity which is represented by that variable. The class **Addends** serves for connecting the expression with its addends. This class contains the same attributes. The attribute *code* stores the code of the variable from the left-hand side of the equation sign in the expression, and the attribute *value* stores the value of the constant

coefficient of the addend.

Although both expressions are of the same type we have to introduce different classes for their representation in order to distinguish them in the process of forming the model and reducing its calculating complexity. We specialize the class **Variable** on the classes **Graph variable** and **Analytic variable**. The class **Analytic variable** does not contain any additional attributes apart from the inherited ones. On the contrary, the class **Graph variable** in addition to the inherited attributes have the two additional attributes *type* and *level*.

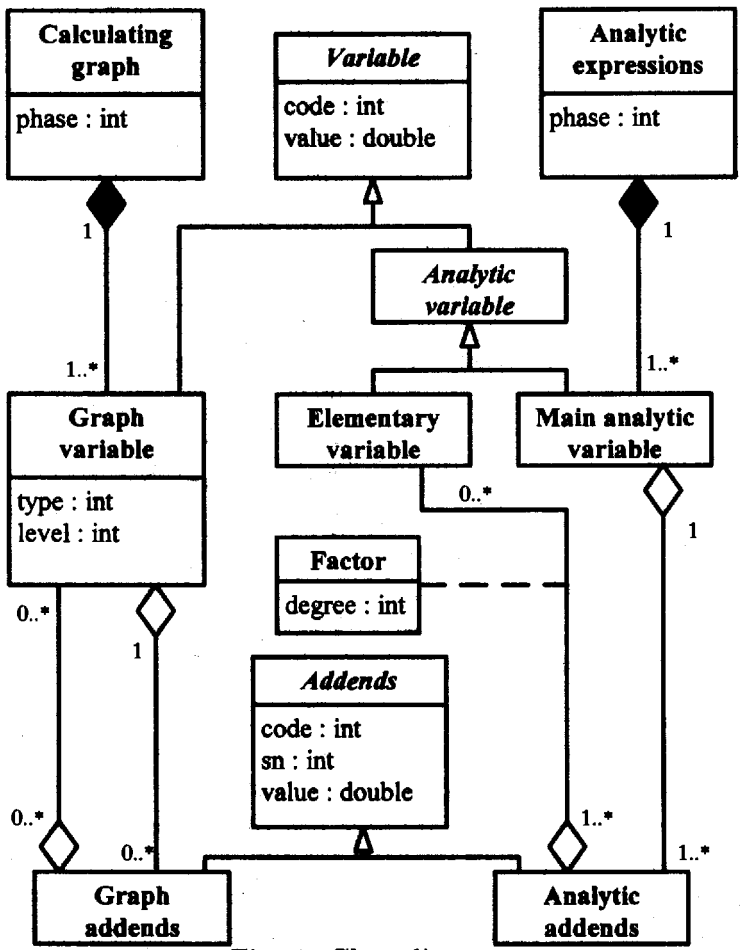

Fig. 1. Class diagram

The attribute *type* takes values from the set {'0','1','2','3'}, where '0'

denotes that the variable has a constant numerical value stored in the attribute *value*, '1' denotes that this variable is one of the robotic mechanism parameters (i.e. mass, moment of inertia,...), while '2' denotes that this variable represents some of the mechanism's generalized coordinates, or the sine or cosine of some of the generalized coordinates. Both the '1' and '2' denote that this variable is not calculated by one of the expressions, because it represents an input quantity. The value '3' denotes that calculation of this variable is described by an expression. The attribute *level* is introduced to enable calculating of the robotic quantities by bottom-up navigating through the database. The variables that are not calculated by means of an expression (variables whose *type* attribute has the values '0', '1' and '2') possess 0 as the value of this attribute; all variables calculated directly via some of the mentioned variables obtain 1 as attribute value etc., depending on the number of steps needed to calculate the given variable.

The class **Addends** is specialized on the classes **Graph addends** and **Analytic addends** because of the same reason of distinguishing the two kinds of expressions. The class **Analytic variable** is the abstract class and it is also specialized on the classes **Elementary variable** and **Main analytic variable** because the fully-developed analytic expressions contain only two kinds of variables. The first are the variables on the left-hand side of the equation sign which are represented by the class **Main analytic variable**, and the second are the variables on the right-hand side of the equation sign, which are really the basic robotic quantities, and which are represented by the class **Elementary variable**. In the expressions for generating the mechanism model there are variables which are on the left-hand side of the equation sign in one expression and on the right-hand side of the equation sign in other expressions, so only one class **Graph variable** is used for representing all the variables from the calculating graph.

First, let us describe storing of the fully-developed analytic expressions. The variable from the left-hand side of the equation sign is stored as an instance of the class **Main analytic variable**. This variable (expression) is connected to its addends by aggregation to the class **Analytic addends**, where the constant coefficients of every addend are stored (in the attribute *value*) so that each addend takes a new instance of the class. Connection of the addend and its factors is represented by the aggregation from the class **Analytic addends** to the class **Elementary variable** in which all basic robotic quantities are stored. Each addend is connected only to its factors, and for every connection we have attached the association class **Factor**,

which represents the degree of that factor stored in the attribute *degree*.

Storing of the expressions in the form of calculating graph is analogous to the previous case. Now the class **Graph variable** contains all the variables (each variable takes a new instance of the class) and represents the both sides of the expressions. At the highest calculation level there are variables which are only on the left-hand side of the equation sign in the expressions. These variables are connected to their addends in a way analogous to the previous case, only the constant coefficients are now stored as instances of the class **Graph addends**. Connection between the addends and their factors is now represented by aggregation from the class **Graph addends** to the class **Graph variable**. Now there is no need for the association class **Factor** because the expressions contain no degrees greater than one. Each factor is an instance of the class **Graph variable** because if it is not a basic robotic quantity, then there is an expression in which it appears on the left-hand side of the equation sign. This expression is again represented by the analogous connections.

In this way all the expressions forming the mathematical model of the mechanism are represented recursively in the form of a calculating graph starting from the variables on the highest calculation level to the basic variables. We use the term *graph* instead of *tree* because some quantities may participate in more than one expression of the quantities from the higher calculation level.

3. Class specification

In the previous paragraph, the class diagram given in Fig. 1, describes only the structure (attributes) of the classes and the relations between them. A detailed specification demands description of the class methods too. In this paragraph, the main classes **Calculating graph** and **Analytic expressions** are specified in more detail in Fig. 2, where the methods of the both classes are specified.

The dynamics of these two classes is described by, the state diagram given in Fig. 3, which specifies the dynamics of the complete system for modelling of the robotic mechanisms. This dynamics is described at the highest level, which means that only the states of the objects which are instances of the classes **Calculating graph** and **Analytic expressions**

are specified. Normally, each state and event of these two objects demands corresponding states and events of objects on the lower level, such as objects of the classes **Main analytic variable, Elementary variable** and **Analytic addends** for the class **Analytic expressions** and objects of the classes **Graph variable** and **Graph addends** for the class **Calculating graph**. These states and events are not described here because of the paper length limitations and because they only additionally complicate the diagram without significant improvement in specification.

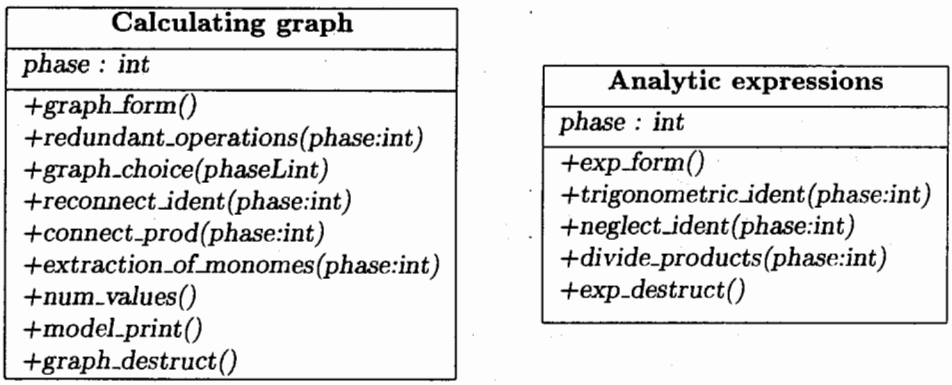

Fig. 2. Class specification

First, the method *graph_form* is started, which on the basis of the method for forming mathematical model of the robotic mechanism dynamics and data on the mechanism's topology and parameters, generates in the database a complete model of the dynamics in the form of a calculating graph. In this paper we adopted the approach of forming the robotic mechanism's mathematical model in closed form according to the Newton-Euler method (Eq. 1). The generated model is an instance of the class **Calculating graph** represented by the state *Starting graph*. The graph nodes represent the quantities participating in the mathematical expressions of the model forming method in a way presented in Fig. 1.

Then, the method *redundant_operations* is applied, which eliminates the redundant operations of the type of multiplying by zero and one and adding zero, and fires the transition to the new object state, *Reduced graph*. In this state the user makes the decision about the subsequent action (do some calculations, exit, or reduce numerical complexity of the graph).

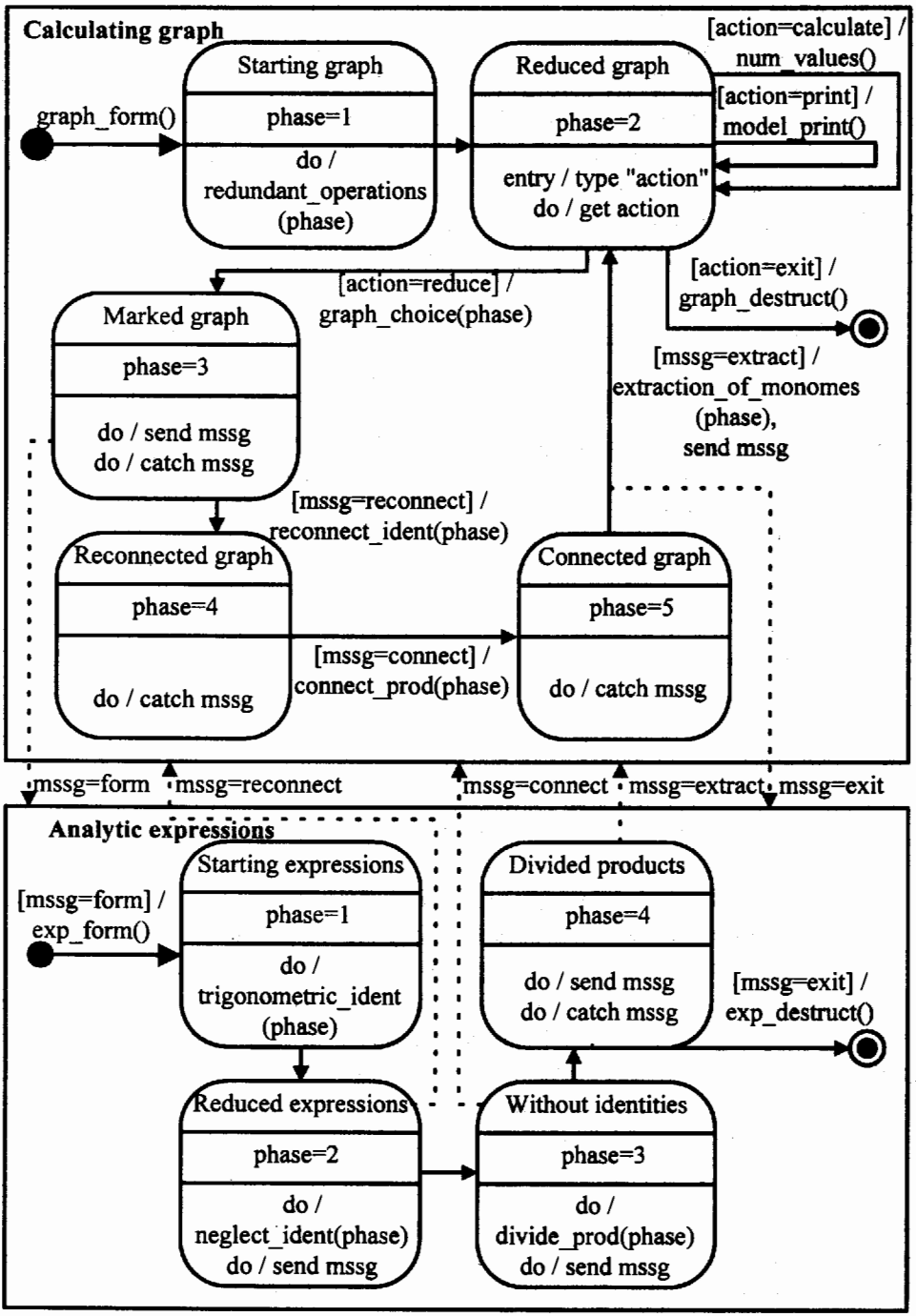

Fig. 3. State diagram

Usually, we want to reduce calculating complexity of the graph before calculations and in this case the method *graph_choice* is performed first, which marks the complete graph or the specific part of the model (subgraph) on which the reducing of the calculating complexity is to be applied. This method also fires the transition to the state *Marked graph.*

Now we need to form the analytic expressions in a fully-developed form for all the quantities of the chosen subgraph (or complete graph), and to store them in the new instance of the class **Analytic expressions**. This is done by the method *exp_form*, which receives the message from the state *Marked graph* and forms a state of the object **Analytic expressions** called *Starting expressions.*

The next step is to use the trigonometric identities in all analytic expressions formed, by which all redundant mathematical operations are eliminated. The method *trigonometric_ident* updates the expressions from the object **Analytic expressions** in order to reduce the number of calculating operations needed for their calculation and fires the transition to the state *Reduced expressions.* Then, the identical analytic expressions are eliminated by the method *neglect_ident*, in order to avoid repetition of the same calculations. If two (or more) identical analytic expressions have been found, one of them is erased from the object **Analytic expressions**, and in the object **Calculating graph** the node which represents the quantity from the left-hand side of the equality sign of the erased expression is disconnected from its subsequent nodes (the quantities which participate in its analytic expression) and connected to the graph node which represents the quantity from the left-hand side of the equality sign in the expression identical to the erased one. This reconnection is done by the method *reconnect_ident*, which is triggered by the message from the state *Reduced expressions.* This method also fires the transition of the object **Calculating graph** to the state *Reconnected graph* and the transition of the object **Analytic expressions** to the state *Without identities.*

To obtain a maximal reduction in the number of mathematical operations the analytic expressions are divided first into the products [4], using the method *divide_prod.* The expressions that were divided are then connected with their products, and these connections are stored in the object **Calculating graph**. This is done by applying the method *connect_prod*, which is triggered by the message from the state *Without identities.* In the same way, the initial expression in the object **Analytic expressions** is sub-

stituted by the expressions into which it has been divided. Then again, on the both objects the transitions to the next states (the state *Connected graph* for the object **Calculating graph** and the state *Divided products* for the object **Analytic expressions**) are fired. Then, on all the expressions from the object **Analytic expressions** the algorithm of extraction of monomes [3] is applied (method *extraction_of_monomes*), by which a new calculating graph is formed. This method is triggered by the message from the state *Divided products* and its application fires the transition of the object **Calculating graph** to the state *Reduced graph* again, and the object **Analytic expressions** is destructed by the method *exp_destruct*, which is triggered by the message from the transition between the states *Connected graph* and *Reduced graph*.

Now the user may choose to reduce complexity of the model again (if he has chosen to reduce the complexity of the subgraph in the first instance), or to perform some output calculations on the generated model. Numerical calculations are carried out on the generated model of the robotic mechanism by the method *num_values*. This method calculates the numerical values for desired model quantities using the values of the generalized coordinates of the mechanism. By bottom-up navigation through the calculating graph the numerical value of each quantity is calculated on the basis of the already calculated numerical values of the quantities participating in the expression of the current quantity. The second output option is the method *model_print* which rewrites from the database a mathematical model of the robotic mechanism with the least possible number of calculating operations. The model is recorded in the data file as a series of mathematical expressions, or as a program in the corresponding programming language. In the end, when the user decided to exit, the method *graph_destruct* is applied which destructs the calculating graph.

4. Conclusion

By means of a detailed analysis of the Newton-Euler method it was established that the mathematical relations forming the robotic mechanism model are in the polynomial form. With the aim of forming mathematical models of robotic mechanisms in symbolic form, it is necessary to develop special data structures, suitable for storing the polynomial expressions and for reduction of their calculating complexity.

In this paper the object-oriented approach is used to design a system for mathematical modelling of robotic mechanisms in symbolic form. Using UML the complete formal specification of the informational requests of complex technical systems can be performed. In the class diagram, consisting part of this modelling language, the statics of the given system is described. The system dynamics is also described using the state diagram, but only at the highest level, which does not reduce the generality of the specification, and yet clearly describes the system. By adding the complete specification of the system dynamics we can obtain a complete formal specification of the given system, which is the basis for its implementation.

References

[1] Vukobratović, M., Potkonjak, V., Dynamics of Manipulation Robots, Theory and Application, Springer-Verlag, 1982.

[2] Vukobratović, M., Borovac, B., Surla, D., Stokić, D., Biped Locomotion, Dynamics, Stability, Control and Application, Springer-Verlag, 1990.

[3] Vukobratović, M., Kirćanski, N., Real-Time Dynamics of Manipulation Robots,Springer-Verlag, 1985.

[4] Racković, M., Generation of the Mathematical Models of Complex Robotic Mechanisms in Symbolic Form, Ph. D. Thesis, University of Novi Sad, 1996.

[5] Racković, M., Surla, D., Closed-Form Mathematical Model of the Dynamics of a Robotic Mechanism with Six Degrees of Freedom, Proc. Second ECPD International Conference on Advanced Robotics, Intelligent Automation and Active Systems, Vienna, Austria, (September 1996), 321-326.

[6] Surla, D., Racković, M., Closed-Form Mathematical Model of the Dynamics of Anthropomorphic Locomotion Robotic Mechanism, Proc. Second ECPD International Conference on Advanced Robotics, Intelligent Automation and Active Systems, Vienna, Austria, (September 1996) pp. 327-331.

[7] Racković, M., Vukobratović, M., Surla, D., Generation of Dynamic Models of Complex Robotic Mechanisms in Symbolic Form, Robotica (1998) volume 16, 23-36.

[8] Booch, G., Rumbaugh J., Jacobson, I., Unified Modelling Language. Version 1.1, Rational Software Corporation, September, 1997, http://www.rational.com.