# Development of a Multibody Dynamics Program Using the Object-Oriented Modeling

Hyung-Suk Han[1,#]

[1] New Transportation System Group, Korea Institute of Machinery & Materials (KIMM), Daejeon, South Korea

## ABSTRACT

A multibody system dynamics analysis program is presented using one of the most useful programming methodologies, the object-oriented modeling. The object-oriented modeling defines a problem from the physical world as an abstract object. The object becomes encapsulated with the data and method. Analysis is performed using the object's interface. It is then possible for the user and the developer to modify and upgrade the program without having particular knowledge of the analysis program. The method presented in this paper has several advantages. Since the mechanical components of the multi-body system are converted into the class, the modification, exchange, distribution and reuse of classes are increased. It becomes easier to employ a new analysis method and interface with other S/W and H/W systems. Information can be communicated to each object through messaging. This makes the modeling of new classes easier using the inheritance. When developing a S/W for the computer simulation of a physical system, it is reasonable to use object-oriented modeling.

**Key Words** : Multibody system dynamics, Object-oriented modeling

## 1. Introduction

Research in the field of multi-body dynamics analysis has been extensively developed over the past 30 years, resulting in the commercialization of multi-body dynamics analysis programs. These analysis programs were developed for the purpose of modeling applications of all kinds of multi-body systems without modification. Some examples of these programs are ADAMS[1], DADS[1], SIMPACK[1] and RecurDyn.[2] However, despite the availability of these commercial programs, it is customary in most cases to use in-house programs instead of these commercial general purpose programs even though it entails a lot more work. There are several reasons for this preference. The first reason is that it is hard for the user to modify the commercial programs to specific requirements. A second reason is that in-house programs simplify input/output from the onset of development to meet the specific requirement. They are also developed considering the user's computer system and communication hardware so they can be implemented effectively. Thirdly, know-how can be supplemented to specific problems. By supplementing design knowledge and know-how with the analysis program, the accuracy of the analytical result can be enhanced and the time necessary for the evaluation can be reduced. The fourth reason is that the user can easily add interface functions since there is a need for integration or interfacing with analysis programs from other fields for multidisciplinary optimization. Thus, in-house programs are very useful in this case.

The multibody system dynamics analysis program, in which the object-oriented modeling is applied, mainly uses the object-oriented language in order to generate the code for equations of motion.[3-11] Their input data format used to construct an analysis model is not a numeric type,

Manuscript received: September 8, 2003 ;
Accepted: October 14, 2003
# Corresponding Author:
Email: hshan@kimm.re.kr
Tel: +82-42-868-7814, Fax: +82-42-868-7844

like in ADAMS and DADS. Sometimes the object-oriented language is used to construct the analysis model. Kunz[12] introduced a numerically efficient program based on four concepts of object-oriented modeling and velocity transformation method. However, since it uses a velocity transformation method, user intervention is required to generate the equations of motion.

This paper presents the application of the object-oriented modeling, one of the best-known programming methodologies, in developing a user-centered multibody system dynamics analysis program. Abstraction, encapsulation, inheritance and polymorphism, which are object-oriented concepts, are applied in the multi-body dynamics modeling and numerical solution process. The equations of motion use generalized equations based on the Cartesian coordinate and adopt object-oriented concepts to develop methods that allow the user or program developer to easily modify or reuse the analysis program. Using the object-oriented language C++, the equations were programmed and applied to an application. As a result, the analysis program described in this paper can be expected to be a user-centered multibody system dynamics analysis program with the advantages of high reusability and good expandability.

## 2. The Object-oriented Modeling

The object-oriented modeling and programming methodology developed from software engineering is already an established software development method. The problem analysis and program implementation process of conventional procedural modeling is operation-oriented. In procedural modeling, the data is taken as an auxiliary factor. On the other hand, in object-oriented methodology, it is implemented based on the objects that are abstracted from the real world in the software domain. Abstraction, encapsulation, inheritance and polymorphism are the characteristics of the objects. The object-oriented programming methodology is currently used in most engineering software applications and is considered to be effective.

Fig. 1 illustrates a multi-body system. If the components of the multi-body system are observed carefully, it is evident that the object-oriented modeling is applicable for the development of multibody system dynamics analysis programs. Abstraction, encapsulation, inheritance and polymorphism, which are characteristics of object-oriented modeling, can be applied to multibody systems. First, the components of multibody systems can be grouped in three categories according to common attributes and procedures. They can be abstracted by categorizing into body, constraint and force. The body has coordinates, mass, moment of inertia and geometrical properties in space. The constraint defines the relative motion of the bodies. The forces represent force and moment that acts on the body. Secondly, it can be seen that the components of multibody systems can encapsulate properties and functions. In the dynamic modeling of multibody systems, detailed information on each object's properties and functions is unnecessary. For example, a translational spring only requires the attached coordinates of two bodies. The objects of the translational spring need to interface only with the attached coordinates. This paper focuses on the advantage of encapsulation to increase the reusability of class and distributional development. In order to define each class, the demand for information on other classes and the main program is minimized. After executing the distributional development, it is possible to independently develop and integrate classes. Thirdly, the functions of objects, which form the equations of motion, can be easily defined. When defining the three types of base classes, the member variables and member functions are defined and then the class is derived using the base class, inheriting the definition of the base class so that it just needs to define the necessary parts without any redefinition. Finally, unlike the FORTRAN-based analysis program, C++ manages memory dynamically so that the concern over size limits of the memory can be reduced. Applying the object-oriented modeling described above to the development of a multi-body dynamics program makes distributional development possible, increases the reusability of the mechanical components's dynamic model, and gives the advantage of being expandable.

## 3. Multi-body Systems Modeling

### 3.1 System equations of motion

The system equations of motion of multi-body systems using the Cartesian coordinate are defined below[13].

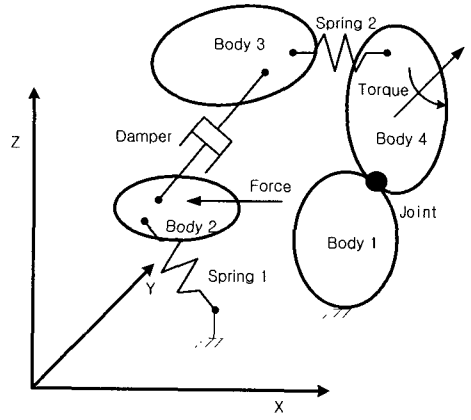


Fig. 1 Multi-body system

Fig. 1 shows a multi-body system composed of body, constraint, joint, force element and control element. In this paper, the Cartesian coordinate and Euler parameter are used. If the multi-body system in Fig. 1 has *nb* number of rigid bodies, then *7× nb* number of coordinates is needed to represent the motion of the system in space. These generalized coordinates are not all independent because of constrains from adjacent bodies. The motion of each body is affected by the kinematic constraint, which defines the link of generalized coordinates. In order to control and understand the motion of multi-body systems, the body, joint and force elements need to be defined in space. If there is a system with *nb* number of bodies, then the generalized coordinate of the system can be defined as Eq. (1).

$$\mathbf{q} = \left[q_1, q_2, \ldots, q_{nb\times 7}\right]^T \tag{1}$$

If the system has *m* number of constraints, it is defined as Eq. (2) and the equation of motion of the constrained mechanical system is defined as Eq. (3).

$$\mathbf{\Phi}(\mathbf{q},t) = \left[\Phi_1(\mathbf{q},t),..,\Phi_m(\mathbf{q},t)\right]^T = \mathbf{0} \tag{2}$$

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{\Phi}_\mathbf{q}^{\ T}\boldsymbol{\lambda} = \mathbf{Q} \tag{3}$$

In order to numerically obtain $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$ that satisfies the constraint given by Eq. (1), the first and second partial differentiations of Eq. (1) need to be solved by the following equations:

$$\mathbf{\Phi}_\mathbf{q}\dot{\mathbf{q}} + \mathbf{\Phi}_t = \mathbf{0} \tag{4}$$

$$\mathbf{\Phi}_\mathbf{q}\dot{\mathbf{q}} = -\mathbf{\Phi}_t \equiv \nu \tag{5}$$

$$\mathbf{\Phi}_\mathbf{q}\ddot{\mathbf{q}} + (\mathbf{\Phi}_\mathbf{q}\dot{\mathbf{q}})_\mathbf{q}\dot{\mathbf{q}} + 2\mathbf{\Phi}_{\mathbf{q}t}\dot{\mathbf{q}} + \mathbf{\Phi}_{tt} = 0 \tag{6}$$

$$\mathbf{\Phi}_\mathbf{q}\ddot{\mathbf{q}} = -(\mathbf{\Phi}_\mathbf{q}\dot{\mathbf{q}})_\mathbf{q}\dot{\mathbf{q}} - 2\mathbf{\Phi}_{\mathbf{q}t}\dot{\mathbf{q}} - \mathbf{\Phi}_{tt} \equiv \boldsymbol{\gamma} \tag{7}$$

Using Eq. (3) and Eq. (7), the system equation of motion in matrix form can be defined as Eq. (8).

$$\begin{bmatrix} \mathbf{M} & \mathbf{\Phi}_\mathbf{q}^{\ T} \\ \mathbf{\Phi}_\mathbf{q} & 0 \end{bmatrix}\begin{Bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \mathbf{Q} \\ \boldsymbol{\gamma} \end{Bmatrix} \tag{8}$$

Here,

$\mathbf{q}$ : *position vector*

$\dot{\mathbf{q}}$ : *velocity vector*

$\ddot{\mathbf{q}}$ : *acceleration vector*

$\mathbf{M}$ : *mass matrix*

$\mathbf{\Phi}_\mathbf{q} \equiv \left[\partial\Phi_j / \partial q_i\right]_{m\times n}$ : *constraint Jacobian matrix*

$\boldsymbol{\lambda}$ : *Lagrange multiplier*

$\mathbf{Q}$ : *generalized force*

$\boldsymbol{\gamma}$ : *right side of constraint acceleration*

In this paper, Wehage's[14] generalized coordinate partitioning method is used as one of the solution methods of Eq. (8).

### 3.2 Class definition

The core of multi-body systems dynamic modeling that uses the object-oriented modeling presented in this paper is the application of class. The class can be considered as a template that defines similar types of objects. A class also has properties. In other words, it has both data and function. The three types of objects described in the previous section have different terms that affect the system equations of motion. Considering these characteristics, the following classes were defined.

#### 3.2.1 Super class

The super class MDObject, which has common attributes and functions of classes that compose multi-body systems, is defined as shown in Fig. 2. The MDObject attributes include the class name, class type, number of DOF, number of constraints, ID of object, ID of instance, DOF pointer, constraint pointer, and others. As for the function, it is composed of initialization, data

process and result output. Here, the id of object, ID of instance, DOF pointer and constraint pointer are the set values determined in the main program. Using the attributes of MDObject the index in Eqs (2)~(8) of each object is determined.

| MDObject |
| --- |
| m_name_object: object name<br>m_name_class: class name<br>m_type_class: type of class<br>m_number_dof: number of DOF<br>m_number_constraint: number of constraint<br><br>m_id_object: id of object<br>m_id_instance: id of instance<br>m_index_dof: index of DOF<br>m_index_constraint: index of constraint |
| Initialize()<br>Read Data()<br>Report() |

Fig. 2 Superclass diagram for multi-body system components

### 3.2.2 Body class

The body class affects the most number of terms in the system equations of motion. If the Euler parameter is used, the body class affects $\mathbf{M}, \mathbf{\Phi}_q, \boldsymbol{\gamma}, \mathbf{Q}$ of Eqs (2), (4)~(7) and Eq. (8). Therefore, the body class can be defined as shown in Fig. 3. However, all the member variables and functions of the body class are not shown in Fig. 3. The member variable and function in Fig. 3 can be queried with previously defined interface and other objects. Objects related to numerical analysis query the member variables to implement the solution process.

| Body |
| --- |
| m_position[7]<br>m_velocity[7]<br>m_acceleration[7]<br>m_force[7]<br>m_mass[7][7]<br>m_jacobian[1][7]<br>m_RHS_velocity[1]<br>m_RHS_acceleration[1]<br>m_constraint[1]<br>m_velocity[1] |
| Calculate mass()<br>Calculate jacobian()<br>Update force()<br>Calculate RHS_velocity()<br>Calculate RHS_acceleration()<br>Calculate constraint()<br>Calculate velocity() |

Fig. 3 Class Body

```
BODY "m_bearing"{
    Method              =: "RYANT";
    Name                =: "m_bearing";
    Origin              =: "0.0,0.0,0.0";
    Mass                =: "0.6";
    Inertia.xx          =: "512";
    Inertia.yy          =: "523";
    Inertia.zz          =: "1022";
    Inertia.xy          =: "5.2";
    Inertia.xz          =: "1.7";
    Inertia.yz          =: "-0.5";
    Fixed.to.ground     =: "False";
    Xg.Force            =: "0.0";
    Yg.Force            =: "0.0";
    Zg.Force            =: "0.0";
    Xl.Torque           =: "0.0";
    Yl.Torque           =: "0.0";
    Zl.Torque           =: "0.0";
    Commet              =: " ";
};
```

Fig. 4 Input data of the class Body

Figure 4 shows the input data of a body class. There are many types of input data but here they are presented simply.

### 3.2.3 Constraint class

The constraint class affects $\mathbf{\Phi}_q, \boldsymbol{\gamma}$ in Eqs (2), (4)~(7) and Eq. (8) in the system equations of motion. Considering this, it can be defined as shown in Fig. 5. The constraint class defines the relative motion among the objects, so the names of the relevant objects and joint coordinates are necessary. Based on this class, a joint can be derived. In this paper, driving constraints are also included in the constraint class. Of course, this categorization can vary depending on the developer and the purpose. The input data format uses the same method as the body class.

### 3.2.4 Force class

The force class affects $\mathbf{Q}$ in Eq. (8) in the system equations of motion. A coordinate is also needed to define the relevant objects and location of the force. Considering this, it can be defined as shown in Fig. 6.

| Constraint |
| --- |
| m_name_body_i<br>m_name_body_j<br>m_triad_i[3][3]<br>m_triad_j[3][3]<br>m_jacobian[6][7]<br>m_RHS_velocity[6]<br>m_RHS_acceleration[6]<br>m_constraint[6]<br>m_velocity[6]<br>m_LagrangeMultiplier[6] |
| Calculate RHS_velocity()<br>Calculate RHS_acceleration()<br>Calculate constraint()<br>Calculate velocity()<br>Calculate reaction_force() |

Fig. 5 Class Constraint

| Force |
| --- |
| m_name_body_i<br>m_name_body_j<br>m_triad_i[3][3]<br>m_triad_j[3][3]<br>m_force[7] |
| Calculate force()<br>Calculate apply_force() |

Fig. 6 Class Force

### 3.2.5 Class hierarchy

Fig. 7 shows the class hierarchy derived from the highest class. The highest class is MDObject, from which the classes such as body, constraint and force class are derived. Using these sub classes, the joint type class, constraint type class and force type class are derived. In the derivation of the class, inheritance and polymorphism are used to easily define a new class. In addition, each class data is managed independently so that a class can be developed independently.

### 3.3 Numerical Analysis

In this paper, a numerical analysis is performed through the application of Wehage's[14] generalized coordinate partitioning method. A different class that implements a numerical analysis was defined and the dynamics analysis was processed in order, as shown in Fig. 8. The class that performs the numerical analysis has vector, matrix and numerical functions in order to derive the system equations of motion. In the process of numerical analysis, the advantages of the object-oriented modeling are well expressed. For dynamics analysis, the derivations of Eqs (2)~(8) are needed and the analysis is performed as shown in Fig. 8. However, the method introduced in this study has the very unique characteristic and advantage of not using common arrays. All data are defined with local variables. In order to explain this in detail, Fig. 9 is provided as an example. As in Fig. 9, in a FORTRAN-based analysis programs, a common array like Fig. 9(a) is used to compose the mass matrix of an object. In order to calculate the mass matrix, each object (e.g. body i) reads the data related to body i from the common array which is to be updated.

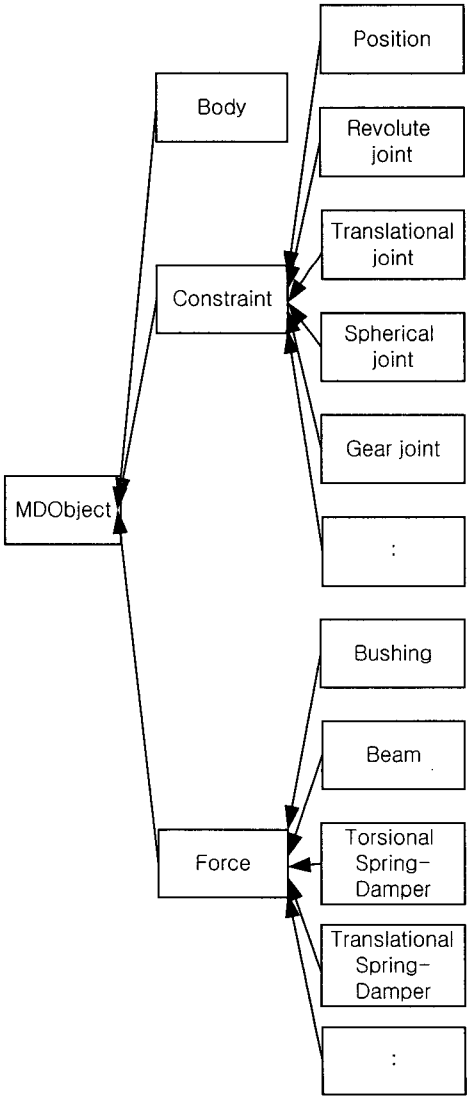


Fig. 7 Multilevel inheritance hierarchy

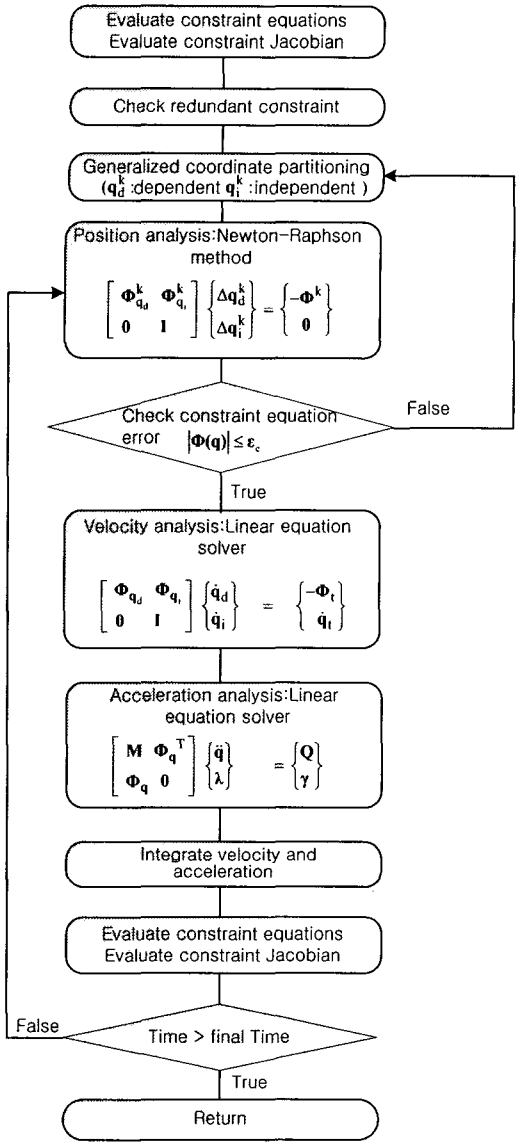


Fig. 8 Dynamic analysis flow

The updated array is used to compose the system mass matrix. Therefore, the index of the array has to be systematically managed. The class developer needs to have knowledge of the index system of the array. On the other hand, in the object-oriented data model used in this paper, each data includes each object's data and has an updated mass matrix. In order to form the system mass matrix, the mass matrix is queried for each body class. Since it does not use a common array, a new class can be defined without any knowledge of the systematic structure of the program. However, the interface is designed with previously defined array names so that the array names used in the interface have to be identical. In the actual development, a new class developer could define a class within a short time and integrate classes into the numerical analysis. The flow diagram shown in Fig. 8 and the components of a multi-body system were made into classes. The derivation of equations of motion and functions related to the numerical analysis were developed using C++ language. The flow of the multi-body dynamics analysis program, O-DYN, is represented in Fig. 10. Currently, O-DYN has developed classes for multi-body dynamics analysis, as shown in Table 1.

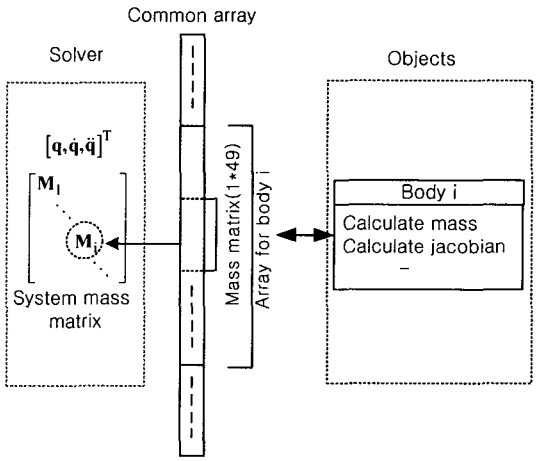


Fig. 9(a) Conventional data model for multi-body dynamic analysis

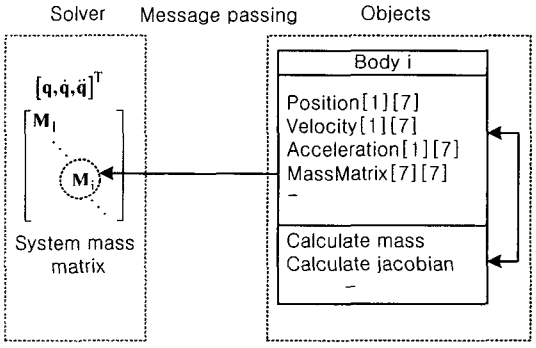


Fig. 9(b) Object-oriented data model for multi-body dynamics analysis

## 4. Application

A dynamics analysis of a reciprocating compressor was conducted using O-DYN, a multi-body dynamics

analysis program developed using the object-oriented modeling presented in this paper. The reciprocating compressor, shown in Fig. 11, is used in refrigerators. The reduction of the compressor vibration is one of the important tasks in advancing the compressor technology.

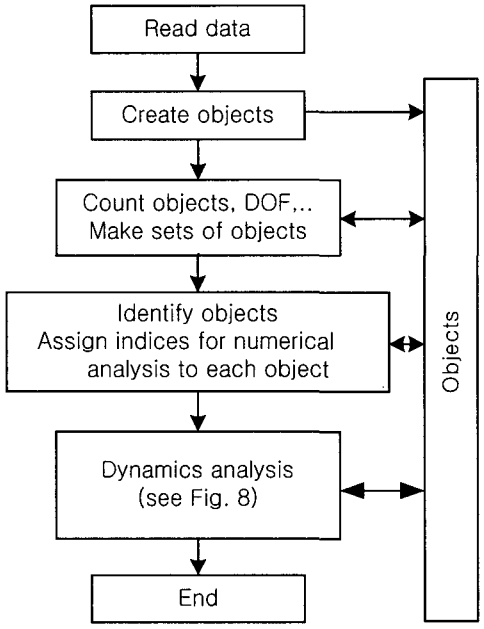


Fig. 10 O-DYN dynamics analysis flow

Table 1 O-DYN classes

| Class type | Classes |
|---|---|
| Body | Rigid Body |
| Constraint | Ground, Position, Point, Revolute joint, Cylindrical joint, Translational joint, Bracket joint, Spherical joint, Rack-and-pinion, Gear joint, Driver |
| Force | Translational-spring-damper, Rotational-spring-damper, Beam, Bushing |

The vibration of a compressor is mainly caused by the pressure fluctuation of a cylinder followed by the motor torque vibration.[15-17] In order to achieve vibration reduction, the dynamic behavior and the exciting force must be predicted from performance for a given condition. In this study, in order to predict the exciting force, the main body named block was fixed to the ground, and the reaction force was evaluated as the exciting force. Fig. 12 shows the compressor and the coordinate system that can be a reference for input/output. Table 2 and Table 3 show the inertia properties of the compressor as shown in Fig. 1 and the multi-body dynamics model. The performance of the motor torque is defined in terms of the angular velocity of the rotor using the performance curve as shown in Fig. 13. The pressure within the cylinder is a function of the rotation angle of the rotor and uses a specific curve obtained from the experiment, which is shown in Fig. 14. The compressor experiences a rotational resistance due to contact with the components. This paper focuses on the gross motion, so based on the results of the experiment it is defined as the damping due to the relative angular velocity of the rotor and the fixed block. The motor angular velocity and variation of the angular velocity were compared with the analysis to obtain the damping coefficient of 2.5 Nms/mm as the rotational resistance. In order to validate the analysis results, a multi-body dynamics analysis program, DADS, was used to perform the same dynamics analysis and the results were compared. The following shows the analysis results of DADS and O-DYN.

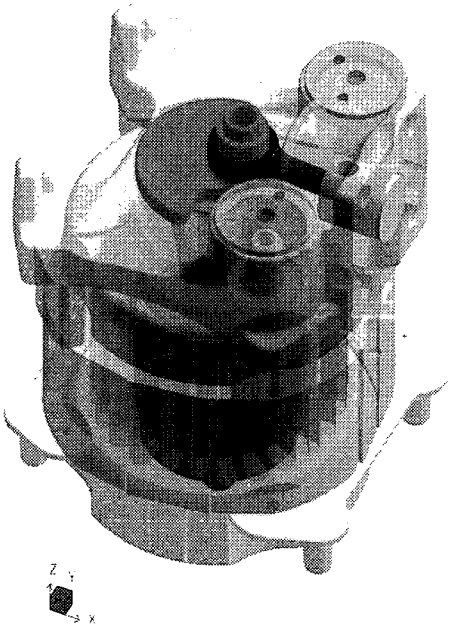

Fig. 11 Reciprocating compressor

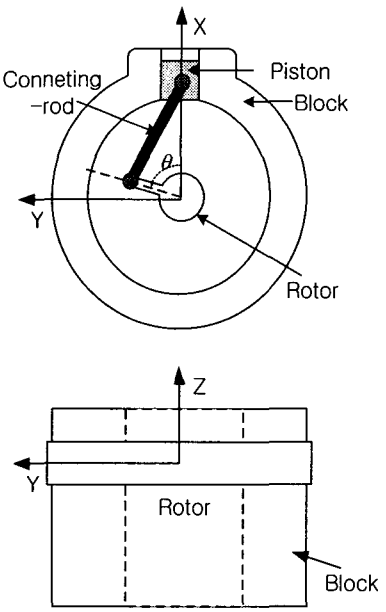


Fig. 12 Schematic diagram for the compressor

Table 2 Inertia properties of the compressor

| Body | Mass(kg) | $I_{xx}$ (kgmm$^2$) | $I_{yy}$ | $I_{zz}$ |
|---|---|---|---|---|
| Block | 5.0 | 20.0 | 0.05 | 15.02 |
| Rotor | 0.04 | 500.0 | 500.0 | 30.0 |
| Connecting-Rod | 0.03 | 1.1 | 10.0 | 3.5 |
| Piston | 0.03 | 3.0 | 3.5 | 4.0 |

Table 3 Dynamic model of the compressor

| Bodies | Block, Rotor, Connecting-rod, Piston |
|---|---|
| Translational joint | Block-Piston |
| Revolute joint | Block-Rotor |
| Revolute joint | Connecting-rod-Piston |
| Cylindrical joint | Rotor-Connecting-rod |
| Motor torque | Figure 13 |
| Compression Pressure | Figure 14 |
| Rolling resistance | Block-Rotor (2.5 Nms/mm) |

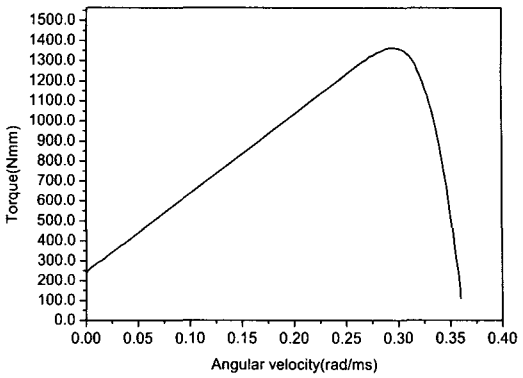


Fig. 13 Characteristics of motor

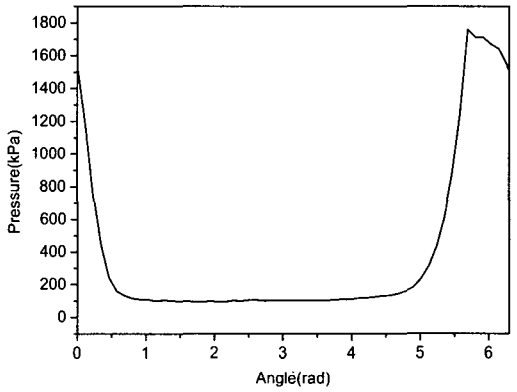


Fig. 14 Pressure versus rotor angle

Figs 15~17 show the x position, velocity and acceleration of the piston. First, it can be seen that the difference in the analysis results of O-DYN and DADS is insignificant. Thus, the reliability of the algorithm and solution of the dynamics analysis program of the object-oriented model could be indirectly validated. The compressor was in motion periodically and the acceleration was 1.03 mm/ms$^2$ at 0° and 1.03mm/ms$^2$ at 180°. In particular, the acceleration had a large variation at the peak point of the piston. In order to reduce the acceleration and the variation rate, the moment of inertia needs to be increased by attaching a weight balancer to the rotating parts. Figs 18~19 show the reaction forces in constraints, which are considered to be the exciting forces. The difference in the results of O-DYN and DADS analyses was also shown to be insignificant. The maximum magnitude of the reaction force is found to be in the x direction, which is 3 times greater than that of the y direction. Using this reaction force, the exciting

force can be predicted according to the compression action and the establishment of a scheme for vibration reduction is required. In addition, a parametric study on the design variables is possible and the design of a lower vibration compressor can be simulated. As described above, the reliability of O-DYN solution was validated through the dynamics analysis of a reciprocating compressor.

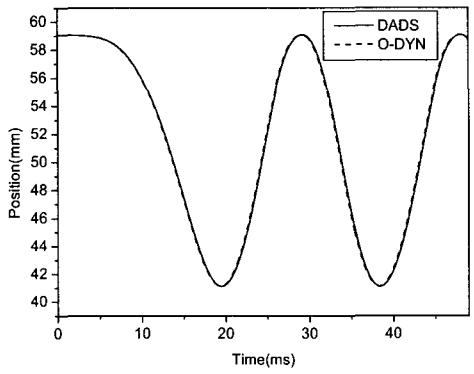


Fig. 15 x position of piston versus time

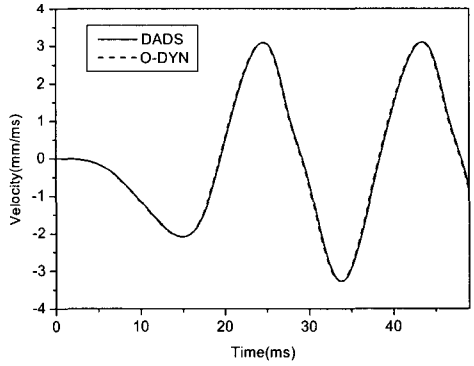


Fig. 16 x velocity of piston versus time

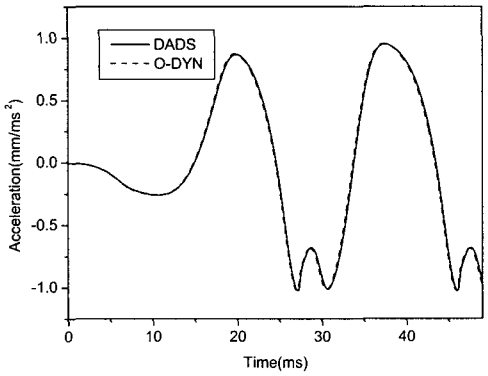


Fig. 17 x acceleration of piston versus time

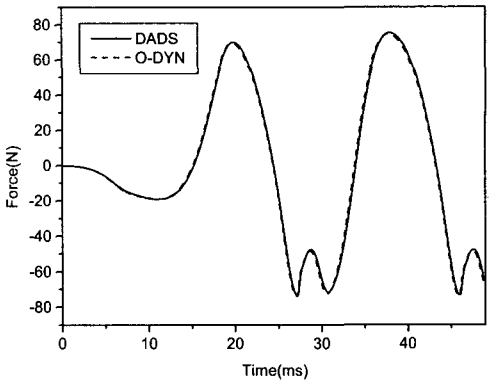


Fig. 18 x reaction force on block at origin

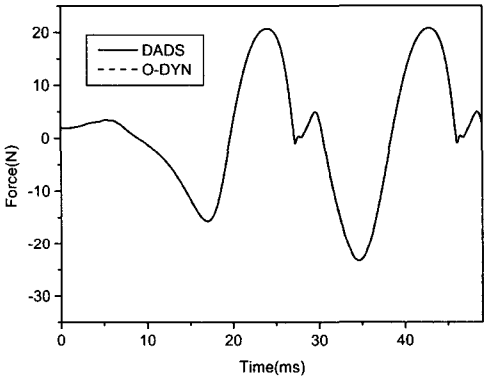


Fig. 19 y reaction force on block at origin

## 5. Conclusion

This paper introduces a design method for a multi-body dynamics analysis program that applies object-oriented modeling. The following advantages of the method were verified. First, the components of multi-body systems were defined as classes and made to be independent so that the modification, convertibility, distributional development and reusability of the components are increased. For example, the distributional development and integration of the developed classes were easily performed without any understanding of the structure of the analysis program. Second, the application of new solution methods and interface with other S/W and H/W are expected to be very easy. This is because the components of the main program and the library pass messages to each object without modification. Third, the new classes can be

easily defined using the inheritance. As a result, the use of object-oriented modeling in developing software for the computer simulation of physical systems is considered to be very effective.

## References


1. Schiehlen, W., Multibody Systems Handbook, Springer-Verlag, Berlin, Germany, 1990.
2. www.functionbay.com
3. Kecskemethy, A., "Sparse-Matrix Generation of Jacobians for the Object-Oriented Modeling of Multibody Systems," Nonlinear Dynamics 9, pp. 185-204, 1996.
4. Otter, M., Elmqvist, H. and Cellier, F. E., "Modeling of Multibody Systems with the Object-Oriented Modeling Language Dymola," Nonlinear Dynamics 9, pp. 91-112, 1996.
5. Tisell, C., Orsborn, K., "Using an Extensible Object-Oriented Query Language in Multibody System Analysis," Advances in Engineering Software 32, pp. 769-777, 2001.
6. Koh, A. S., Park, J. P., "Object-Oriented Dynamics Simulator," Computational Mechanics 14, pp. 277-287, 1994.
7. Kecskemethy, A., Hiller, M., "An Object-Oriented Approach for an Effective Formulation of Multibody Dynamics," Computer Methods in Applied Mechanics and Engineering, Vol. 115, pp. 287-314, 1994.
8. Sreenath, N., "A Hybrid Computation Environment for Multibody Simulation," Mathematics and Computers in Simulation 34, pp. 121-140, 1992.
9. Hocke, M., Seybold, J. and Ruhle, R., "Data Models and Simulation of Mechanical Systems," Simulation Practice and Theory 4, pp. 319-333, 1996.
10. Kecskemethy, A., Lange, C. and Grabner, G., "Object-Oriented Modeling of Multibody Dynamics Including Impacts," ECCM-2001, Cracow, Poland, 2001.
11. M. Anantharaman, "Flexible Multibody Dynamics – An Object-Oriented Approach," Nonlinear Dynamics 9, 205-221, 1996.
12. Kunz, D. L., "An Object-Oriented Approach to Multibody Systems Analysis," Computer and Structure 69, pp. 209-217, 1998.
13. Haug, E. J., Computer Aided Kinematics and Dynamics of Mechanical System, Allyn and Bacon, USA, 1989.
14. Wehage, R. A., Haug, E. J., "Generalized Coordinate Partitioning for Dimension Reduction in Analysis of Constrained Dynamics Systems," Journal of Mechanical Design, Vol. 104, No. 1, pp. 247-255, 1982.
15. Padhy, S. K., "On the Dynamics of a Rotary Compressor : Part 1-Mathmatical Modeling," Advances in Design Automation, ASME, Vol. **1**, pp. 207-217, 1993.
16. Padhy, S. K., "On the Dynamics of a Rotary Compressor : Part 2-Experimental Validation and Sensitivity Analysis," Advances in Design Automation, ASME, Vol. 1, pp. 219-227, 1993.
17. Yangisawa, T., Mori, M., Shimizu, T. and Ogi, T., "Vibration of a Rolling Piston Type Rotary Compressor," International Journal of Refrigeration, Vol. **7**, No. 4, pp. 237-244, 1985.