Diaz-Calderon, A. / Towards a Unified Representation of Mechanisms for Robotic Control Software, pp. xx - yy
International Journal of Advanced Robotic Systems, Volume y, Number x (200x), ISSN 1729-8806

Towards a Unified Representation of Mechanisms for Robotic Control Software

Antonio Diaz-Calderon, Issa A. D. Nesnas, Won S. Kim and Hari Das Nayar
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA
({first.last}@jpl.nasa.gov)
OphirTech, Altadena, CA, USA,
(hari@ophirtech.com)

Abstract: *This article gives an overview of the Mechanism Model paradigm. The mechanism model paradigm provides a framework to modeling mechanisms for robotic control. The emphasis is on the unification of mathematical models of kinematics/dynamics, geometric information and control system parameters of a variety of robotic systems (including serial manipulators, wheeled and legged locomotors), with algorithms needed for typical robot control applications.*
Keywords*:unified mechanism model, real-time control, serial manipulator, wheeled mechanisms, kinematics, dynamics.*

1. Introduction

The work presented in this paper describes 1) a unified approach for modeling the geometric and dynamic properties of a variety of robotic systems and 2) generic algorithms that operate on the model to perform common analyses needed in the control of such systems. The goal of this work is to provide a common representation for describing mechanical systems including robotic arms and wheeled and legged vehicles and provide built-in algorithmic functionality (for example, forward and inverse kinematics and collision detection) needed for typical robot control applications.
The implementation of this approach will provide the Coupled Layer Architecture for Robotic Autonomy (CLARAty) (Volpe, Nesnas, Estlin, Mutz, Petras & Das, 2001), (Nesnas, Wright, Bajracharya, Simmons & Estlin, 2003), (CLARAty, 2005) on-board software with a more generic infrastructure for mechanism modeling and analysis. The modeling software covers mobility mechanisms, robotic arms, rover masts, and mechanical legs. The modeling software will provide the necessary information for real-time computation of kinematics, dynamics, and collision prediction.

1.1. Motivation

Current implementations of kinematic and geometric descriptions of mechanisms have different representation paradigm, which makes sharing kinematic and dynamic information very difficult. Furthermore, it is desirable for some applications to treat the mechanism as a whole body (for instance when dealing with rover-arm coordination algorithms). Some other applications may require the treatment of appendages as separate elements. For instance when controlling a rover and a mast with panoramic cameras.
A modeling paradigm that supports both views of mechanism control and that provides for simplicity of information sharing
A unified modeling approach has the following advantages:

- Allows re-use of kinematic/dynamic and geometric information for a variety of robotic systems. Particular implementations will be instiated by reading in model files that specify detailed description of a specific robotic system.
- Provides centralized storage for managing model information. This includes creation, deletion, update, extension and reconfiguration of the mechanical models.
- Ensures consistency of the model information for use by multiple algorithms. This simplifies the integration of algorithms into the software architecture.

- Reduces duplication in model representation between rover mobility and manipulation software.
- Enables the development of generic algorithms for forward, inverse, and differential kinematics. In the absence of specialized versions, the generic algorithms provide out-of-the-box functionality.
- Supports specific implementations to override generic algorithms whenever appropriate for optimal performance.
- Enables the verification of specialized kinematics algorithms against their generic counterparts.
- Supports a variety of data input models such as Denavit-Hartenberg and zero configuration.

This article contains software requirements for developing a unified mechanical model. It also contains requirements for algorithms and describes the interaction of the models with the rest of the on-board robotic software. In this paper, the term "mechanism" refers to any mechanical system and does not imply a closed loop mechanical chain.

1.2. Related work

Attempts at creating a unified representation for mechanism are limited. The approaches found in the literature can be used to describe robotic systems from a number of system attributes, e.g., kinematic, geometric etc.

The most notable approaches are the DARTS/Dshell (Jain, 1991), (Rodriguez, Kreutz-Delgado & Jain, 1991), the Open Robot Control Software Kinematics Pagkage (OROCOS, 2005), the Operational Software Components for Advanced Robotics (OSCAR, 2005), (Kapoor & Tesar, 1998), RoboML (RoboML, 2005), (Makatchev & Tso, 2000), ORCA (ORCA, 2005), (Brooks, Kaupp, Makarenko Williams & Oreback 2005) and the Nucleus robotic control toolkit (Nucleus, 2005).

DARTS/Dshell is a high-fidelity dynamics simulator that models the motion of flexible multi-body systems under internal and external interactions. It has been used to model robotic systems and spacecraft. Applications include hardware-in-the-loop testing and off-line simulations. In the sequel we will show how DARTS/Dshell' model representation is used in the mechanism model context.

The original goal of the OROCOS effort was to develop open source software for robotics applications. It has since branched into two separate developments:

- Open Real-time Control Services for real-time control applications
- Open Robot Control Software that provides class libraries and a framework for robot applications.

The Kinematics Package in OROCOS is at the design concept stage and no software has been developed. It is intended to address more general mechanical systems and not be restricted to tree-topology systems. While the objectives of the Kinematics Package have been documented, the approach to be used for its implementation has yet to be clearly defined. Furthermore, due to its open source nature and lack of funding the future of the Kinematics Package in OROCOS is unclear. As a result, the OROCOS Kinematics Package currently exists as a collection of objectives without a detail description of its approach or implementation.

OSCAR provides utilities in the form of libraries for performing computations needed in analysis, real-time control or simulation of manipulators. In addition to math utilities, it contains algorithms for performing generic forward and inverse kinematics, motion planning and dynamics. OSCAR offers many alternative options in its operations. For example, for motion planning, trajectories can be generated using trapezoidal, spline or motion blending algorithms. OSCAR currently appears to allow only models of serial-chain manipulators. OSCAR's primary application is robotics education. While OSCAR provides generic software utilities for robot arms (serial-chain manipulators) the approach presented in this paper models more general kinematics systems.

1.3. Problem statement

The CLARAty architecture is a unified and reusable software framework that provides robotic functionality and simplifies the integration of new technologies on robotic platforms. Being a domain-specific robotic architecture its objective is to operate a number of heterogeneous mobility platforms with different physical capabilities (including serial kinematic chains ; e.g., manipulators, masts) and hardware architectures.

The heterogenous narture of the mechanisms supported by CLARAty introduces the need to have a unified representation covering mechanical and geometric properties as well as algorithms needed for typical robot control applications.

Furthermore, to enable CLARAty developers/users to build new manipulators and mobility platform models more efficiently in a more systematic manner for an entire system, there is a need to come up with a unified geometric, kinematic, and quasi-static representations of mechanisms supporting serial kinematic chains, wheeled locomotors, multi-legged locomotors, contact control, geometric models for collision detection, generic forward/inverse kinematics, and unified model specification inputs.

Although the body of work presented in the literature covers different aspects of our problem (e.g., serial kinematic chains and real-time control), there is no approach that covers all the types of mobilitry mechanisms covered in CLARAty. Moreover, eventhough the work presented in the literature provides algorithms for robot control applications, they all seem to lack a very important property, that of geometric representation, to allow the integration of collision detection algorithms. This functionality is desired in

particular when the robot is expected to operate fully autonomous and be capable to generate safe commands.

1.4. Approach

To provide a unified representation of mechanisms, which supports heterogeneous mobility platforms with different physical capabilities, the mobility and manipulation group in CLARAty has developed a new approach to modeling of robotic systems. The approach presented in this paper, called *Mechanism Model*, is a software architecture that provides a unified representation for robotic systems, which can be used to execute real-time control of the mechanism. The unified representation is based on the DARTS/Dshell approach. The primary differences between the Mechanism Model in CLARAty and DARTS/Dshell are:

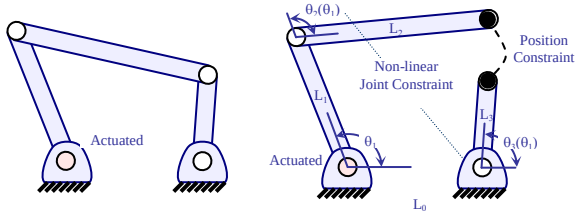
- The modeling in DARTS/Dshell is geared for high fidelity simulations while the Mechanism Modeling of CLARAty is geared for real-time high-frequency control and planning. The latter also supports overriding of generalized solutions with specialized ones.
- DARTS/Dshell uses much more detailed models (body flexibility, actuator and transmission modeling, etc.,) which are needed for high-fidelity simulations.
- Simulation software and control software solve complementary problems (e.g. simulation software solves the forward dynamics while controls software solves the inverse dynamics).
- DARTS/Dshell makes extensive use of recursive algorithms while Mechanism Model uses corresponding iterative algorithms, which are more amenable for on-board flight implementations.

This article is organized as follows: Section 2 provides the general requirements the software architecture must address. Section 3 presents the design of the mechanism model paradigm. Section 4 summarizes the types of constraints handled by the architecture. Section 5 explains the model data input. Section 6 summarizes the generic algorithms available in the architecture. Section 7 presents the relationship of this paradigm in the CLARAty framework.

2. General requirements

In the design of the Mechanism Model paradigm, we separated mechanism models (e.g., kinematics, dynamics, etc.) from mechanism control to allow client software to use and test the kinematics and dynamics of the mechanisms independent of the hardware. Furthermore, it was required that the Mechanism Model supports the following computations for the mechanism model:

- Kinematics computations—forward, inverse, and differential kinematics.



(a) Four-bar mechanism

(b) Four-bar mechanism modeled with either (i) position constraint, or (ii) with non-linear joint constraints

Figure 1. Closed loop kinematic chains

- Quasi-static computations of forces and torques considering: joint flexibility (stiffness), gravity force and other applied forces, gravity deflection.
- Environmental contact constraints—position, force, torque, and stiffness.
- Resolution of multiple simultaneous kinematics constraints.
- Collision detection.

Initially, full dynamics computations (inertial forces) will not be supported. However, models will support future extensions for dynamics computations.

2.1. Mechanism types

The types of mechanisms that can be described in the Mechanism Model paradigm include:

- Serial manipulators—multi-degree of freedom robotic arms, masts, and legs (e.g. a 5-dof arm with a turret gripper carrying multiple instruments).
- Simple closed-chain mechanisms—four- and six-bar planar mechanisms (Figure 1).
- Wheeled locomotors—multi-wheeled mechanisms with different drive and steering configurations (Figure 2). This includes fully-steerable (e.g. Rocky 8's six-wheel drive six-wheel steering rocker bogie mechanism), partially-steerable (e.g, Rocky 7's all-wheel drive front wheel steering mechanism), and skid-steerable mechanisms (e.g. ATRV's).
- Legged locomotors—multiple legs attached to a body, e.g. LEMUR robot (Figure 3).
- Composite mechanisms—any combination of the above types (e.g. Phoenix lander and MER rover Spirit and Opportunity (Figure 4)).

In contrast to these types of mechanisms the mechanism model software will not directly handle parallel and hybrid kinematic structures. However, it is possible to model a simple parallel structure as a tree topology by breaking the closed chain and solving for the closed chain using joint or position constraints (Figure 1).

(a) Rocky 8 rover (b) Rocky 7 rover (c) ATRV

Figure 2. Different mobility platforms

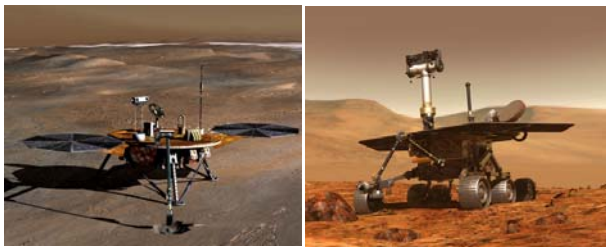
Figure 3. Legged robot

3. Mechanism model design

A mechanism is represented as a tree topology in which an arbitrary number of rigid bodies are connected to one another via joints. The tree topology captures the geometric relationships between all elements in the mechanism such as sensors and bodies, and serves as a repository of mechanical model information, which includes fixed (non-articulated) transformations, joint constants that do not depend on articulation values and component geometry. As a result, the proposed tree representation is stateless: position, velocity, and acceleration information relative to an inertial frame is not stored in the mechanism model. This is an important feature because it will enable various system states to be updated at different rates and enable the use of different parts of the tree at a time. It will also allow algorithms to use the mechanism model tree to predict future states for any given input state. The trade that is made here is the cost of re-computing derived states vs. making copies of mechanism model for each client application and keeping all their internal state up to date.

From the Mechanism Model perspective there is a single inertial frame, which acts as the root of the tree. This is important for creating composite mechanisms from identical components. For example, if we have a six-legged robot with identical legs, we only need a model for one of the legs and we can then create the robot by inserting the leg model at the mounting point for each leg.

Geometric information (for example for use in collision detection) is kept in the same model tree and it is hierarchically defined in terms of the bounding shape information of the physical component (Figure 5).



(a) Phoenix lander (b) MER rover

Figure 4. Composite mechanisms

The geometric description of a body is defined in a manner consistent with the bounding shape proposed by Leger (Leger, 2002). The following material is adapted from the cited work for the sake of completeness of the exposition.

In general, a body contains a bounding shape tree (see Figure 5.) that describes containment relationships among the geometric objects of a single body. The bounding shape tree describes different levels of granularity at which the geometry of the object is described, having the finest bounding shape resolution at the leaves of the tree.

Bounding shapes are represented either as 2D or 3D shapes (e.g. we represent terrain surfaces and walls by 2D open meshes and manipulator links by 3D shapes such as cylinders, boxes, spheres, and/or convex hulls).

4. Constraint modeling

There are two types of constraints handled by the Mechanism Model: *Joint* and *Cartesian* constraints. Joint constraints are constraints that couple a joint to another through a linear or nonlinear relationship (e.g. $q_i = f(q_k)$, where q is a generalized joint coordinate). Cartesian constraints are further divided into: *end effector* and *contact* constraint. Contact constraints are used to specify the desired surface contact between two frames, while end effector constraints are used to specify the desired absolute or relative position of a frame.

5. Model data input

Mechanism model parameters are specified in an eXtensible Markup Language (XML) (The World Wide Web Consortium, 2005) input file (refer to (Nesnas, Kim, Nayar & Diaz-Calderon, 2005) for a detailed description of the XML model data format). Each mechanism or appendage is defined in a separate XML file and a complete mechanism model is assembled by reading in multiple XML files (e.g. for a mars rover we need arm model, mast model, and mobility model are stored in separate files).

The mechanism model file defines necessary mount points by name to attach appendages defined in separate model files.

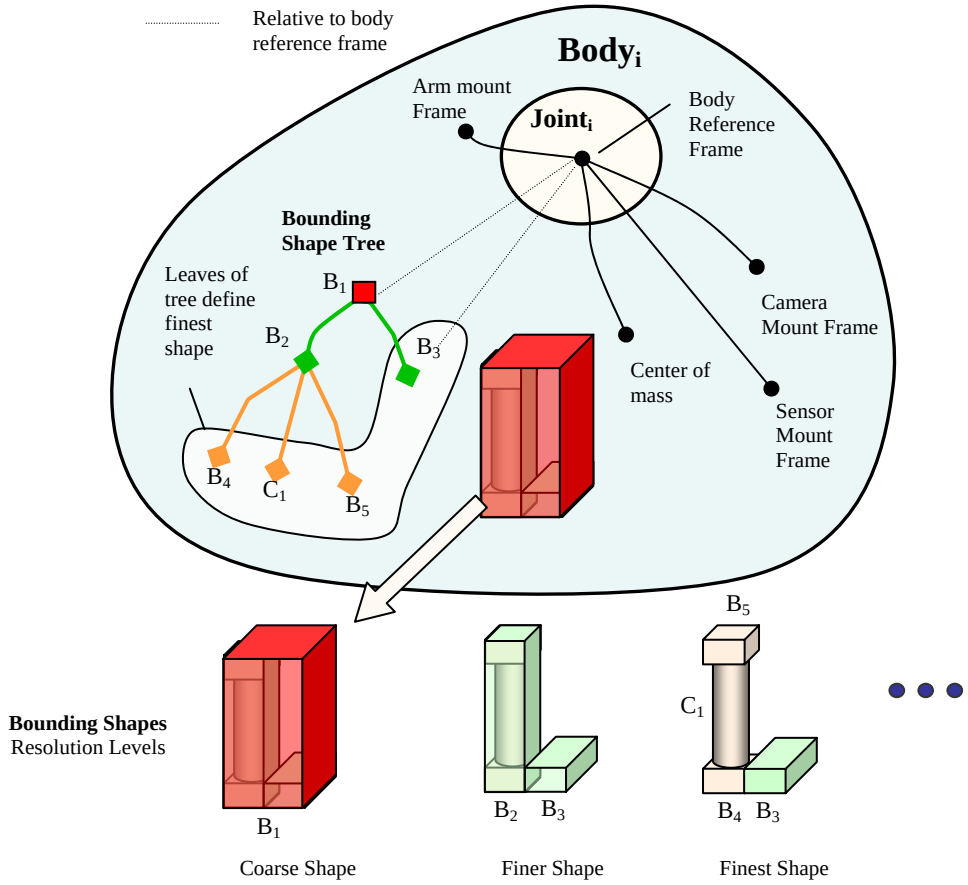


Figure 5. A mechanical component an its bounding shape tree

The input parameter file supports the required kinematic parameters based on the kinematic representation selected as well as supporting the definition of center of mass, inertia matrix, bounding shape tree. A number of kinematic representations are supported including homogenous transform, zero position (also known as product of exponentials) (Murray, Li & Sastry, 1994), Denavit-Hartenberg where the rerference frame of the *i-th* body is located at the *i+1-th* joint (Paul, 1983), (Spong, & Vidyasagar, 1989), and Denavit-Hartenberg with Craig's modification where the reference frame for the *i-th* body is located at the *i-th* joint (Craig, 1989).
Regardless of the representation selected, model parameters are converted to an internal representation, which is similar to that used in DARTS/DShell. Specifically, we treat each body as having a single joint that ties the body to its parent in the mechanism model tree. This approach is general enough to enable modeling of mechanisms of various types and simplifies the software structure (**Figure 6**).
The choice of coordinate frames on the bodies is not based on DARTS/Dshell but are consisten with the following conventions

- For single degree-of-freedom joints, the z-axis is aligned with the articulation axis.
- For multiple degrees-of-freedom joints, the z-axis is aligned with at least one joint axis.
- The body reference frame is located at the center of rotation of the revolute joints.

6. Algorithms

A set of built-in algorithms needed for typical robot control applications will be implemented in the software package. These include generic forward and inverse kinematics algorithms and collision detection between objects in the model. These will be extended in the future to include gravity compensation and dynamics analysis. Facilities will be provided to allow users to by-pass the generic built-in algorithms with more-efficient customized specific implementations.

6.1. Forward kinematics algorithm

The built-in forward kinematics algorithm to determine relative positions and orientations between coordinate frames in the model will be performed by accumulating relative poses between successive frames in the tree using tree-traversal iterators to navigate between frames on the tree. The user-interface to the software will allow users to query a frame in the tree for its pose relative to any other frame in the tree.

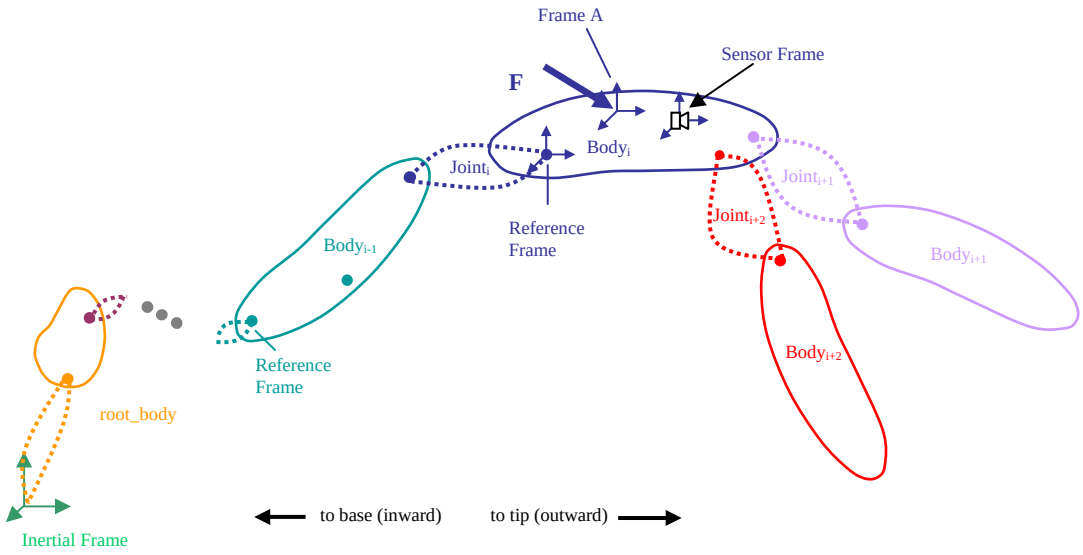


Figure 6. Mechanism model: bodies and joints

6.2. Inverse kinematics algorithm

A built-in generic inverse kinematics algorithm will be implemented using a numerical approach. *Constraint managment* software is used to administer the generic solution for inverse kinematics problems based on the type of Cartesian constraints. The solution of a general set of Cartesian constraints that simultaneously apply to the kinematic system, is found through an iterative numerical approach (e.g., Newton-Raphson). In this approach the constraint manager sets up the Cartesian constraint vectors to solve for, and then uses a constraint solver to determine the configuration of the kinematic system that best solves for the set of Cartesian constraints. This approach will handle multiple simultaneous constraints.

6.3. Collision detection algorithm

The initial implementation of the collision detection algorithm in this software package will be to adapt Leger's approach (Leger, 2002). In this approach, the geometry of each physical object is defined with a hierarchy of composites of bounding shapes. At high levels in the hierarchy, one or a few geometric objects are used to bound the physical object. At lower levels in the hierarchy, a more refined geometric model is possible by using many smaller objects (see Figure 5). The efficiency of this algorithm is obtained by checking for collisions between objects at high levels of the hierarchy and penetrating to deeper levels only when a collision is found to occur between high-level objects.

Although currently there is one collision detection algorithm, the framework is general enough to support multiple collisiton detection algorithms operating on the bounding shape tree.

7. Interfacing mechanism model with CLARAty control abstractions

CLARAty provides a number of control abstractions including locomotion and manipulation abstractions. The mechanical model for the components defined by these abstractions is expressed in terms of the mechanism model. In this context the mechanism model can be used either as a stand-alone abstraction for kinematic analisys, or as part of the control softrware for the robotic system. When the mechanism model is used to describe abstractions such as manipulator or locomotors interface abstractions are defined. These interface abstractions and their corresponding control abstractions will enable the mechanical sub-system to be controlled independently. For example, a system with a rover and a manipulator arm can be treated as two independent control systems by utilizing the Manipulator_Model and Wheeled_Locomotor_Model interfaces for the arm and the rover respectively (Figure 7). Alternatively, one can use the Mechanism_Model anstraction to simultaneously coordinate the arm with the rover motions.

8. Acknowledgements

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract to the National Aeronautics and Space Administration. The authors would like to acknowledge and thank Anne Wright, Raymond Cipra, Max Bajrachara and Daniel Clouse for their contributions.

9. References

Brooks, A., Kaupp, T., Makarenko, A., Williams, S. & Oreback, A. (2005). Towards Component-Based Robotics, Principles and Practices of Software

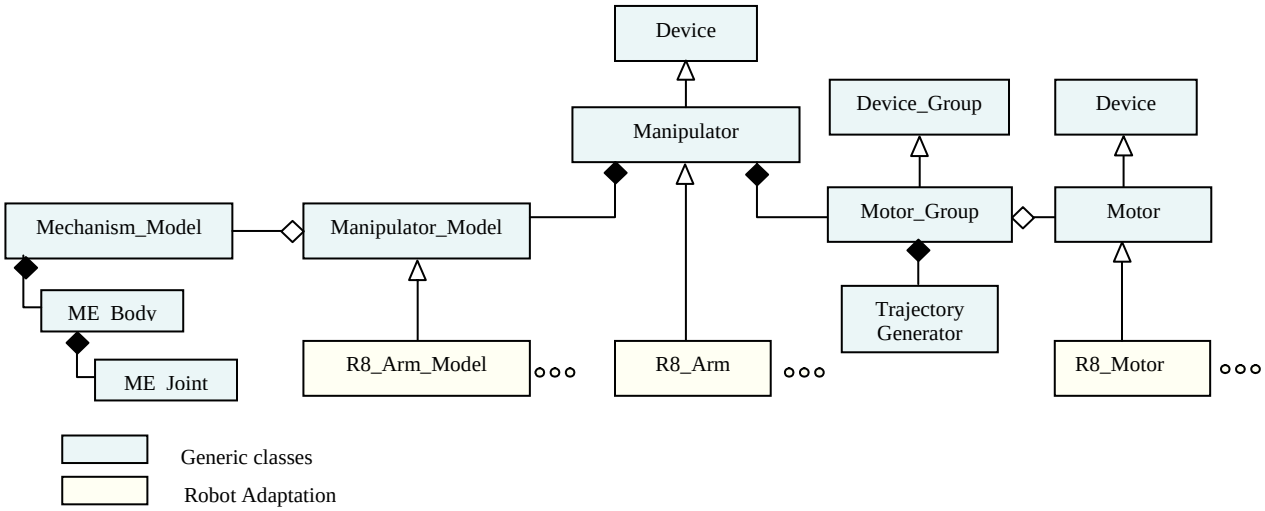


Figure 7. Example of using Mechanism_Model with Manipulator control classes. Similar structures will be used for Wheel_Locomotor and Legged_Locomotor classes.

Development in Robotics (SDIR2005), ICRA2005 Workshop, Barcelona, Spain, April 2005.

Craig, J. J. (1989). *Introduction to robotics, mechanics and control*, Addison-Wesley, Reading, Massachusetts.

CLARAty (2005), http://clarity.jpl.nasa.gov

Jain, A. (1991). Unified Formulation of Dynamics for Serial Rigid Multibody Systems, *Journal of Guidance, Control and Dynamics*, vol. 14, pp. 531-542.

Kapoor, C. & Tesar, D. (1998). A reusable operational software architecture for advanced robotics, Proceedings of the Twelfth CSIM-IFToMM Symposium on theory and Practice of Robots and Manipulators, Paris, France, July 1998.

Leger. C. (2002). Efficient Sensor/Model Based On-Line Collision Detection for Planetary Manipulators, Proceedings of 2002 ICRA.

Makatchev, M. & Tso, S. K. (2000). Human-Robot Interface Using Agents Communicating in an XML-Based Markup Language, Proceedings of the 2000 IEEE International Workshop on Robot and Human Interactive Communication, Osaka, Japan, September 2000.

Murray, R. M, Li. Z & Sastry, S. S. (1994). *A Mathematical Introduction to Robotic Manipulation*, CRC Press, Boca Raton.

Nesnas, I.A., Wright, A., Bajracharya, M., Simmons, R. & Estlin, T. (2003). CLARAty and Challenges of Developing Interoperable Robotic Software, invited to International Conference on Intelligent Robots and Systems (IROS), Nevada, October 2003.

Nesnas, I.A., Kim, W.S., Nayar, H.D. & Diaz-Calderon, A. (2005). *CLARAty: Mechanism model software design document*, JPL Technical Report No. NNNNN, Available from http://clarity.jpl.nasa.gov

Nucleus (2005). http://www.energid.com/site/nucleus.htm

ORCA (2005). http://orca-robotics.sourceforge.net/index.html

OROCOS (2005). http://www.orocos.org/

OSCAR (2005). http://www.robotics.utexas.edu/rrg/research/oscarv.2/

Paul, P. R. (1983). *Robot manipulators: Mathematics, programming, and control*, MIT Press, Cambridge, Massachusetts.

RoboML (2005). http://www.roboml.org/

Rodriguez, G., Kreutz-Delgado, K. & Jain, A. (1991). A Spatial Operator Algebra for Manipulator Modeling and Control, *The International Journal of Robotics Research*, vol. 10, pp. 371-381.

Spong, M. W. & Vidyasagar, M. (1989). *Robot dynamics and control*, John Wiley & Sons.

Volpe, R., Nesnas, I.A.D., Estlin, T., Mutz, D., Petras, R. & Das, H. (2001). The CLARAty Architecture for Robotic Autonomy, Proceedings of the 2001 IEEE Aerospace Conference, Big Sky Montana, March 2001.

The World Wide Web Consortium (2005). Extensible Markup Language (XML), htttp://www.w3.org/XML