Structure-Based Computational Modeling Architecture for Robotics

Abhinandan Jain[1]

Abstract— **We describe a computational architecture for meeting a diverse range of robot modeling needs encompassing analysis, simulation and embedded modeling for robotic systems. The architecture builds upon the spatial operator algebra theoretical framework for computational dynamics. It allows applications to meet the broad range of computational modeling needs coherently and with fast, structure-based computational algorithms. The paper describes the SOA computational architecture, the DARTS computational dynamics software, and application modeling layers.**

I. INTRODUCTION

Computational models pervade all aspects of robotics, from the design and analysis of systems, to use within onboard planning and control architectures for system development and test, and during their operation. Recent DARPA programs such as Autonomous Robotics Manipulation (ARM) and the DARPA Robotics Challenge (DRC) are focused on increasing autonomous robot operation in unstructured environments. Autonomous manipulation and mobility requires increased sophistication and robustness, and this in turn increases modeling needs for manipulation and control, grasp design and analysis, task planning, legged locomotion, online calibration etc [1].

Such robotics demands have stimulated the development of physics-based simulators to support development and testing [2–5]. However significant challenges remain and the DRC program is investing in the development of simulators for use by the robotics community. Challenges in computational modeling include the large variety of modeling requirements, the complexity of the numerical algorithms, designing models that adequately capture the physics, obtaining correct parameters for seeding the models, and obtaining fast computational performance.

The challenges are even greater for models used within autonomy software. Such embedded models support task and motion planning, state estimation in the presence of noise and uncertainty and real-time closed-loop control. Computational speed requirements are significantly higher and the types of information needed span a much broader range. Thus while a simulator is required to do one function well, i.e., simulate the system state time history, embedded models have to perform several functions well.

Due to their inherent complexity, embedded computational models are often platform specifid, over-simplified or point solutions to meet a narrow a range of functions or tailored to the needs of specific autonomy modules. Low-fidelity and fragmented models degrade the overall system performance. Fragmentation requires extra measures to ensure the consistency of modeling assumptions and expectations as well as to keep changing model parameters and data in sync across the system modules. In addition, there is the added cost of implementing, testing and validating these special purpose models.

In this paper we describe a computational modeling architecture for robotic systems that has been designed to overcome these challenges for robotics analysis, simulations and embedded modeling. Our approach is to develop a highly capable modeling layer to serve as a foundation for the diverse and demanding needs of application layers. The architecture is illustrated in Figure 1. Its lowest layer is the *spatial operator algebra (SOA)* theoretical framework for computational mechanics [6]. SOA provides mathematical tools for expressing, analyzing and computing a very broad range of robot mechanics quantities. It's expressiveness allows it to coherently meet the large variety of mechanics modeling needs for robot models. It uses spatial operators for the concise mathematical description and analysis of dynamics quantities, as well as the generation of fast, structure-based computational algorithms. Section II provides an overview of the SOA framework, and Section II-A describes the SOA processes for developing low-order, structure-based computational algorithms.

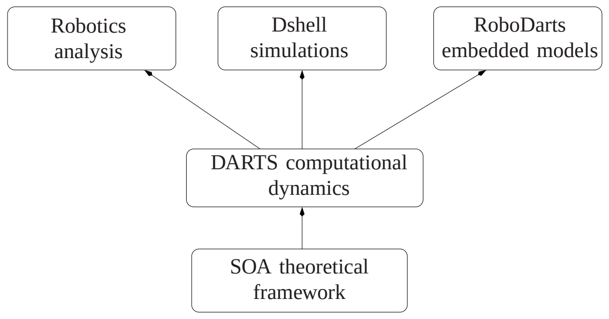
Robotics analysis
Dshell simulations
RoboDarts embedded models
DARTS computational dynamics
SOA theoretical framework

Fig. 1. Overall robotics computational mechanics architecture built upon the SOA theory, the DARTS software library for analysis, simulation and embedded modeling applications.

The next layer of the architecture is the *DARTS* computational dynamics C++ library whose design is based upon the SOA framework. DARTS provides methods for the computation of a broad range of robot modeling quantities and is described in Section III. The DARTS library also includes a number of dynamics solvers needed for time simulation of robot dynamics. These solvers utilize the underlying DARTS methods and differ in the type of dynamics they handle, as

[1]A. Jain is with the Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109, USA `Abhi.Jain@jpl.nasa.gov`

well as the algorithms they use. These solvers are described in Section III-A.

The robot modeling applications build upon the SOA and DARTS layers. In Section IV we discuss models for supporting robotics analysis. In particular we detail the *PyCraft* toolkit for the development, testing and maturation of advanced computational dynamics techniques. Section V turns to the important area of dynamics simulation. DARTS and its solvers form the heart of the *Dshell* simulation framework for the development of system level physics-based simulations. Dshell provides ways to organize and manage the large number of component models in these simulations and to reuse these component and sub-system models across simulations. Dshell has been adapted to develop rover and flight dynamics system simulations. Finally, in Section VI we describe the *RoboDarts* embedded modeling layer that is built upon DARTS. RoboDarts is designed for use by autonomy software to meet the diverse modeling needs for planning, estimation and real-time control. Specific attention is paid to performance speed, time variability of the tasks and environment and the robot platform's interactions with them.

II. SOA Theoretical Framework

The *spatial operator algebra (SOA)* theory and mathematical framework for multibody dynamics [6] has been developed over two decades of research. SOA provides a mathematical language for succinctly characterizing and analyzing complex dynamics quantities for articulated robotics systems. It also provides a natural avenue for developing fast, structure-based computational dynamics algorithms. These features make possible its use as a powerful architecture for the computational mechanics of robotic systems.

SOA makes use of *minimal coordinate* system representations. Thus inter-link hinge motion is parameterized by minimal set of coordinates instead of with redundant coordinates subject to constraints. This reduces the size of the equations of motion and avoids the need to manage constraint violation errors required with redundant coordinates. A side-effect of using minimal coordinates is that the mass matrix, and other system matrices, are dense and configuration dependent and thus, more complex. However, these quantities have rich structure that the SOA provides the mathematics tools to handle.

The mapping between joint and link velocities serves as a simple example to introduce the notion of *spatial operators*. The 6-dimensional spatial velocity $\mathcal{V}(k)$ of the k^{th} link depends on that of its parent link's spatial velocity $\mathcal{V}(k+1)$ via the relationship $\mathcal{V}(k) = \phi^*(k+1,k)\mathcal{V}(k+1) + H^*(k)\,\dot{\theta}(k)$, where $\phi^*(k+1,k)$ denotes the rigid body propagation matrix for the pair of links, $H^*(k)$ characterizes the permissible hinge motion across the connecting hinge and $\theta(k)$ the k^{th} hinge coordinates. This component level relationship can be converted into a system level relationship of the form $\mathcal{V} = \phi^* H^*\,\dot{\theta}$, where $\mathcal{V}$ and $\dot{\theta}$ are system level vectors obtained by stacking up the vector contributions from each link. ϕ and H are block matrices whose elements are defined by the component $\phi^*(k+1,k)$ and $H^*(k)$ link-level matrices. ϕ and H are examples of *spatial operators*.

The concise $\mathcal{V} = \phi^* H^*\,\dot{\theta}$ relationship reflects the rich structure of the ϕ operator. Indeed, for tree/serial topology systems, $\phi = (I - \mathcal{E}_\phi)^{-1}$, where $\mathcal{E}_\phi$ is another spatial operator whose structure is closely related to the adjacency matrix for the graph associated with the connection topology of the linkages [7]. $\mathcal{E}_\phi$ is always nilpotent for serial/tree topology systems.

The mathematical structure of the spatial operators make possible several transformations and simplifications of dynamics quantities. A seminal example of this is the system mass matrix, $\mathcal{M}(\theta)$. It has been shown to have the factored form $\mathcal{M}(\theta) = H\phi M \phi^* H^*$, where M is a block-diagonal spatial operator with link spatial inertias. While this factorization involves non-square factors, the following sequence of analytical spatial operator expressions with alternative factorizations involving square factors. and an expression for its inverse, can be derived [8]:

$$\begin{aligned} \mathcal{M} &= H\phi M \phi^* H^* \\ &= [I + H\phi\mathcal{K}]\mathcal{D}[I + H\phi\mathcal{K}]^* \\ [I + H\phi\mathcal{K}]^{-1} &= I - H\psi\mathcal{K} \\ \mathcal{M}^{-1} &= [I - H\psi\mathcal{K}]^* \mathcal{D}^{-1} [I - H\psi\mathcal{K}] \end{aligned} \tag{1}$$

These expressions involve the additional ψ, $\mathcal{D}$, and $\mathcal{K}$ spatial operators described in [6]. The analytical expressions in Eq. 1 are remarkable in that they hold generally for arbitrary size serial/tree topology systems. They are direct consequences of the intrinsic mathematical structure of the spatial operators. Indeed, the $\mathcal{E}_\phi$ and ϕ operators are special instances of a broader family of *spatial kernel operator (SKO)* and *spatial propagation operator (SPO)* operators [6]. The ψ operator is also an instance of an SPO operator.

It has been shown using graph theory ideas that spatial operator expressions, such as in Eq. 1, generalize to the case where the component bodies and hinges are flexible [6]. Remarkably, they hold even when the system graph is transformed into simpler graphs by aggregating groups of bodies into new variable geometry compound bodies. This observation forms the basis of *constraint embedding* techniques that transform non-tree topology graphs into tree graphs, and thus extend the applicability of Eq. 1 to even non-tree topology systems [6].

Spatial operator analysis applies to a number of robot dynamics problems such as joint space dynamics, operational space dynamics, under-actuated systems, sensitivity analysis, diagonalized dynamics etc. The SOA provides a unified way to tie together results and analysis obtained from disparate approaches, and to understand the relationships among them. Also, the SOA has been used for several novel analyses and results not possible by other means [6]. The rich structure of spatial operators arises, perhaps not coincidentally, from the close mathematical parallels with concepts and analysis techniques developed in the optimal estimation arena.

A. Structure Based Computational Algorithms

We now examine the intimate connections between spatial operator expressions and low-order computational algorithms that is one of the hallmarks of the SOA-based computational architecture. As a case in point, we observed earlier the equivalency between the link-level velocity recursive relationships (and the implied computational procedure) and the $\mathcal{V} = \phi^* H^* \dot{\theta}$ spatial operator expression. The connection between spatial operator expressions and recursive computational procedures is in fact much broader and deeper. Thus, while a spatial operator expression of the form $\phi^* x$ may suggest the need for a matrix/vector product for evaluation (quadratic order cost), it turns out that such a product can always be evaluated via a linear-order recursive *scatter* algorithm that starts at the base of the tree and traverses the links towards the tip bodies as shown on the right of Figure 2. The recursive algorithm for computing the link velocities is a special case of such a scatter algorithm. Similarly, dual spatial operator expressions such as ϕx do not

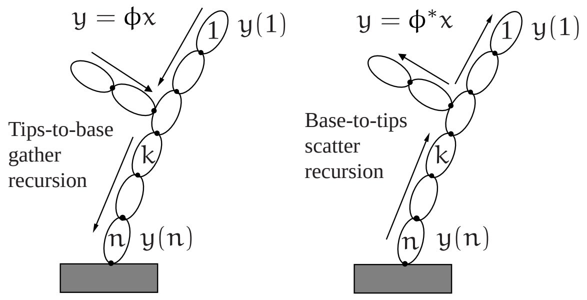

Fig. 2. Tips-to-base gather and base-to-tips scatter recursions to evaluate ϕx and $\phi^* x$, respectively for tree-topology systems.

require expensive matrix/vector products either, but instead can always be evaluated using a linear-order, recursive *gather* algorithm that starts at the tips and traverses the links towards the base body while accumulating results from converging branches. An example application of a gather computation is the computation of compensating torques for an end-effector force. The value of this torque is $\mathcal{J}^* \mathfrak{f}_e$, where $\mathcal{J}$ is the end-effector Jacobian and $\mathfrak{f}_e$ the end-effector spatial force. The spatial operator expression for the Jacobian is $\mathcal{J} = \mathcal{B}^* \phi^* H^*$, and thus the torque $\mathcal{J}^* \mathfrak{f} = H \phi \mathcal{B} \mathfrak{f}_e$ which can be directly computed using a recursive gather sequence.

The recursive path of the gather and scatter algorithms are isomorphic to the topological structure of the system, and hence we refer to them as *structure-based* algorithms. As the system topology changes, the gather and scatter recursion paths change accordingly so that the computations remain correct. Such structure-based algorithms always exist for spatial operator expression involving SKO and SPO spatial operators such as $\mathcal{E}_\phi$, ϕ, ψ etc.

Inverse dynamics computations require evaluation of the $\mathcal{M}(\theta)\ \ddot{\theta}$ product. While direct evaluation is expensive and requires the explicit evaluation of the $\mathcal{M}$ mass matrix, we can instead use the spatial operator expression $\mathcal{M}$ in Eq. 1 to note that this product has the form $H \phi M \phi^* H^*\ \ddot{\theta}$. This expression can be evaluated using a scatter recursion for $M \phi^* H^*\ \ddot{\theta}$ followed by a gather recursion on the result to compute the full product. This procedure is of linear cost and does not require the explicit evaluation of $\mathcal{M}$. It is essentially the well known *Newton-Euler inverse dynamics algorithm* [9].

Forward dynamics computations require the evaluation of $\mathcal{M}^{-1}(\theta)\mathcal{T}$. Once again, the cost of this evaluation can be reduced to linear-order by using the spatial operator expression for $\mathcal{M}^{-1}$ from Eq. 1 so that we need to evaluate $[I - H\psi\mathcal{K}]^* \mathcal{D}^{-1} [I - H\psi\mathcal{K}]\mathcal{T}$. From simple examination we see that this can be evaluated using a gather recursion followed by a scatter recursion without ever explicitly requiring $\mathcal{M}$ or $\mathcal{M}^{-1}$. This procedure is in fact the well known linear cost *articulated body (AB) forward dynamics* algorithm [8, 10].

The above examples illustrate the ability to convert spatial operator expressions for dynamics quantities into low-order, structure-based computational algorithms from simple examination of the expression structure. Other more advanced examples of such mapping also extend to (Lyapunov) spatial operator expressions involving matrix/matrix products such as $\phi X \phi^*$ and the dual $\phi^* X \phi$ spatial operator expressions. The former appears within the mass matrix spatial operator expression in Eq. 1. Expressions such as these can also be computed using low-order scatter and gather algorithms based on a operator decomposition of such products into a block diagonal matrix, and off diagonal terms that can be computed from the block diagonal part. We refer the reader to [6] for additional details. The well-known *composite-rigid body algorithm* for the mass matrix [11] and the *articulated body inertia algorithm* [8, 10] are both examples for the first Lyapunov quadratic form. The latter quadratic form appears in the *operational space inertia* spatial operator expression $\mathcal{B}^* \psi^* H^* \mathcal{D}^{-1} H \psi \mathcal{B}$, and forms the basis for the fastest available computational algorithm for its evaluation [6]. For additional examples of such spatial operator to structure-based algorithms we refer the reader to reference [6]. The key take away messages are that not only is it possible to obtain fast computational methods from analytical spatial operator expressions, but that a large and diverse family of such fast algorithms emerge naturally from the SOA approach.

III. DARTS COMPUTATIONAL DYNAMICS

As described above, the SOA framework provides us with a mathematical vocabulary and tools that cover a broad spectrum of dynamics analysis needs and fast algorithms for computing them. We have used the SOA architecture to develop the *Dynamics Algorithms for Real-Time Simulation (DARTS)* computational dynamics C++ software. Earlier generations of DARTS implemented the SOA rigid and flexible body forward dynamics algorithms for spacecraft simulation. More recent versions of DARTS more completely exploit the richer structure-based SOA computational architecture. Thus DARTS is not designed for a specific function, eg. dynamics simulation, but rather encompasses the broad family of SOA based methods that can be combined to meet different modeling needs.

Since relative pose, velocity and acceleration computations permeate dynamics computations, DARTS includes a generic layer for frame to frame queries. The frames layer provides methods to obtain relative data for any pair of frames defined in the system. Lazy evaluation and data caching is used for on demand computation and to avoid recomputing unchanged values. This abstraction layer considerably simplifies link-level computations. The frames can be specialized to become links, viewing cameras, or other moving elements such as planetary bodies.

DARTS provides classes for bodies and hinges that connect them. The bodies are organized as a directional graph (digraph) consisting of a spanning tree, and additional bilateral constraints that may be present. Bilateral constraints are themselves defined by hinges that characterize the permissible motion across the constraints. A variety of hinge types with 0 to 6 degrees of freedom are available to choose from. These classes help to define the physical model of the system.

Computational algorithms operate on user defined subgraphs of the physical graph. Thus an inverse dynamics computation can be limited to a part of the system by defining a subgraph for the subsystem. The SOA scatter/gather structure-based algorithms traverse just the links in the subgraph. When the subgraph happens to be the full system, the inverse dynamics for the full system is computed. The ability to work with subgraphs is very useful for systems such as mobile and multi-arm platforms, where intermediate information for sub-systems is often needed for planning and control. The lazy evaluation and data caching approaches for frames are used for dynamics computations as well. Thus velocity changes do not require recomputation of configuration dependent quantities.

DARTS includes implementations of all the SOA scatter/gather algorithms described in Section II such as for inverse dynamics, articulated body forward dynamics, composite body inertias, composite momentum, operational space inertias etc. The forward dynamics algorithm actually is a mixed dynamics variation of the standard AB forward algorithm, in that it allows the accelerations for arbitrary coordinates to be prescribed, i.e., specified as inputs, and the algorithm solves for the required hinge torques. The mixed dynamics is handled by a simple modification of the standard AB gather/scatter sweeps. DARTS also has methods to iteratively solve for coordinates consistent with constraints on the system that can be used for inverse kinematics computations.

DARTS also supports treating a subgraph of bodies as a variable geometry compound body. Such an aggregation step can be used to eliminate loops from the graph topology of the system as needed for constraint embedding dynamics. All of the SOA algorithms are applicable to the transformed subgraph.

DARTS allows the run-time addition and deletion of bodies as well as their reattachment. Being structure-based, the SOA algorithms continue to work as is after such changes to the system topology. Geometrical shapes can be attached to the bodies. The geometry information can be used by collision detection and 3D visualization modules as needed. Interfaces exist for the automatic synchronization of the DARTS state with such support modules.

A. *DYNAMICS SOLVERS*

Physics-based simulations are important application problems for dynamics modeling. Simulations use solvers to compute time trajectories of the system state by solving and propagating the equations of motion which have the form

$$\begin{aligned} \mathcal{M}(\theta)\ \ddot{\theta} + \mathcal{C}(\theta,\ \dot{\theta}) &= \mathcal{T} \\ G_c(\theta, t)\ \dot{\theta} - \mathfrak{U}(t) &\geqslant 0 \end{aligned} \tag{2}$$

$\mathcal{C}(\theta,\ \dot{\theta})$ in the first expression represents the Coriolis, gyroscopic and gravitational terms in the equations of motion for the tree-topology part of the system. The second equation defines any additional constraints that may be present. The constraints for whom equality holds are referred to as (smooth) bilateral constraints while the inequality ones are non-smooth unilateral constraints (eg. contact constraints). Key factors effecting the quality of a solver are its computational speed, modeling fidelity, and numerical accuracy. Solvers can be categorized as being smooth or non-smooth solvers. Non-smooth solvers can handle impulsive collision and contact dynamics. DARTS implements several solvers that build upon the family of dynamics methods and algorithms available within DARTS. A brief overview of the key solvers is described next.

Tree smooth dynamics solver: This solver is for the smooth dynamics of tree topology systems with no additional constraints. The underlying minimal coordinate equations of motion are ordinary differential equations. This solver uses the optimal low-order AB mixed dynamics algorithm for solving the equations of motion together with ODE integration schemes for the time propagation of the system state.

Tree-augmented (TA) smooth constrained dynamics solver: For systems with non-tree topologies, the smooth dynamics must handle constraints on the system dynamics. The TA solver does so in multiple steps: (a) use the mixed AB dynamics algorithm for solving the unconstrained equations of motion; (b) use this solution to solve for the unknown constraint forces; and (c) correct the solution from (a) with the accelerations from the constraint forces from (b). Steps (a)-(c) provide a solution for the constrained equations of motion. Since the equations of motion use redundant coordinates, a DAE integrator is required to integrate the state derivatives. Also a constraint error stabilization scheme is required to keep the solution on the constraint manifold. While steps (a) and (c) use the fast mixed AB algorithms, step (b) requires the computation of a Schur complement matrix of the form $G_c\mathcal{M}^{-1}G_c^*$ whose size depends on the constraints. While this can be an expensive matrix to compute, we have used SOA techniques to show that its structure is closely related to that of the operational space inertia, and thus linear cost SOA algorithms can be used to

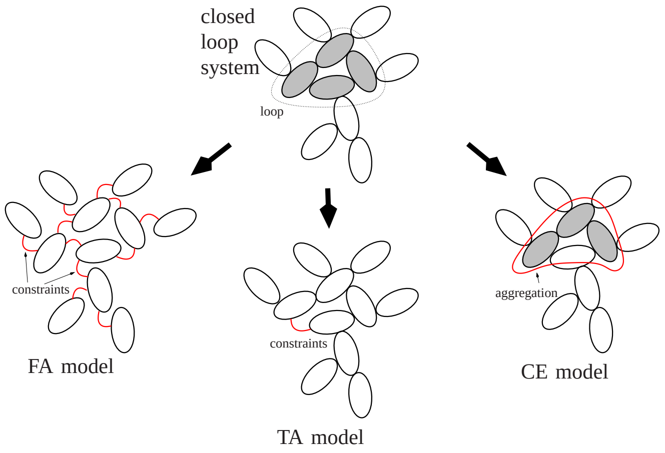

Fig. 3. Dynamics solver options for time simulation of smooth, constrained dynamics problems.

evaluate the Schur complement and significantly reduce the cost of step (b) [6].

Fully-augmented (FA) smooth constrained dynamics solver: The FA solver is similar to the solvers in dynamics packages such as [2, 4, 12]. It can be a viewed as a limiting case of the TA solver, where all the hinges are replaced by constraints. The system dynamics consists of a set of independent bodies subject to a large set of constraints. Since no hinges are involved, the equations of motion are simple to set up. The matrices involved are large, but sparse. The solution technique is similar to that for the TA solver, except that steps (a) and (c) are trivial, and the bulk of the cost lies in step (b). Again, a DAE solver and constraint error stabilization is required.

Constraint-embedding (CE) smooth constrained dynamics solver: Unlike the TA and FA solvers that impose constraints on the unconstrained dynamics, the CE solver eliminates the constraints from the system graph using constraint embedding techniques [13]. In this approach, links in loops with constraints are aggregated into compound bodies to eliminate the constraints from the transformed graph. After eliminating all constraints in this manner, the transformed graph is a tree, and the tree dynamics solver can be used. This technique is faster and also numerically more robust because only minimal coordinates are integrated. Moreover, the underlying dynamics is an ODE thus permitting simpler integrators and larger time steps. There are extra computations involved in the scatter/gather traversals across compound bodies, but these additional costs are far outweighed by the savings from using the tree dynamics solver.

Non-smooth contact/collision dynamics solver: Contact and collisions involve impulsive, non-smooth dynamics. DARTS implements a complementarity based algorithm for solving such non-smooth dynamics with support for elastic and inelastic collisions. [14, 15]. The DARTS approach differs from prior approaches [4] in its use of a minimal coordinates formulation that is significantly faster and avoids problems from the use of redundant coordinates. This solver handles both bilateral and unilateral constraints. An example of the dynamics simulation of a tire changing scenario is shown in Figure 4. The DARTS based models are used within embedded models in the autonomous execution [1] of the same scenario shown on the right side of the figure.

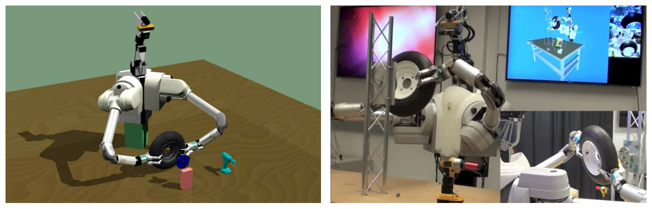

Fig. 4. The left figure shows the time simulation of the non-smooth dynamics of a tire changing robotic task using a DARTS based simulator. The figure on the right shows the same scenario being executed autonomously with DARTS based embedded models.

A comparison of the computational costs of the TA, FA and CE smooth constrained dynamics solvers is described in [16]. The CE solver is the fastest, followed by the TA solver, with the FA solver being a distant third. A comparison of the computational cost of the DARTS non-smooth solver and the traditional non-smooth solver schemes is described in [14, 15]. The DARTS solver is significantly faster and has the advantage of using minimal coordinates. For smooth constrained dynamics, the TA and the CE solvers can be combined, where constraint embedding is used to eliminate constraints associated with small loops, while the TA augmented approach is used for constraints associated with larger loops.

IV. APPLICATION: ANALYSIS

Robot analysis is a broad ranging area with uses spanning robot design and optimization (for actuator sizing, mass properties, kinematic characteristics), robot performance analysis (workspace, singularity characterization), robot operations (designing sequences, gait analysis, determining operational constraints) etc. One of the intended uses of DARTS is as an analysis tool for such robotics problems. Python bindings for all the C++ classes and methods in DARTS are available to facilitate such use. The Python bindings are auto-generated using the SWIG tool [17] and faithfully mirror the underlying C++ classes. The Python bindings allow users to exercise the full C++ functionality within the Python scripting environment. This includes the running of Python analysis scripts as well as interactive user sessions. A typical session begins with the initialization of the DARTS robot model via a model description file, followed by various analysis computations from the command line or via analysis scripts and supplementary Python modules. The Python interface allows users to leverage Python's extensive built-in capabilities, and its large collection of open source extension modules. Furthermore, being a programming language, it is possible to use variables, loop and other constructs within scripts. The scripting capability removes the burden of building or compiling software from users. Parametric analysis is straightforward using this infrastructure.

We have also developed the *PyCraft* environment [18] to allow the development, validation and benchmarking of dynamics algorithms. PyCraft extends DARTS by adding C++ classes that mimic SOA operators. This allows users to create operator instances for their robotic system, and combine the operators in different ways via intuitive expressions that mirror the underlying SOA expressions. Once again, while all the PyCraft classes can be accessed at the C++ level, the Python bindings allow the classes to be instanced and manipulated within Python scripts and sessions.

Thus, PyCraft exploits the succinct and expressive nature of SOA operators to allow the easy evaluation of complex dynamics quantities. For example, the $\mathcal{T} = H\phi\mathcal{M}\phi^* H^* \ddot{\theta}$ computation for inverse dynamics can be computed from the PyCraft Python prompt via the following statement:

```
>>> T = H*Phi*M*PhiStar*HStar*thetaddot
```

The SOA forward dynamics spatial operator expression $\ddot{\theta} = \mathcal{M}^{-1}\mathcal{T} = [I - H\psi\mathcal{K}]^*\mathcal{D}^{-1}[I - H\psi\mathcal{K}]\mathcal{T}$ can be evaluated via

```
>>> thetaddot = (I-H*Psi*K).transpose()*
                     D.inverse()*(I-H*Psi*K)*T
```

Similarly complex quantities such as the mass matrix and its inverse can be evaluated using the SOA expressions for them. The $+$, $*$ etc binary operators are overloaded to invoke the appropriate low-order scatter/gather SOA algorithms for the operation. The PyCraft extension makes available virtually the complete suite of SOA operator based analysis at the command line. This is useful for developing, testing and maturing complex and new algorithm ideas in an easy to use environment. The operator layer reduces the analysis burden by allowing work to be carried out at a much higher level without getting swamped by the low level details of typical dynamics formulations. This enables rapid prototyping of new computational ideas and algorithms for robot dynamics.

V. APPLICATION: SIMULATION

Section III-A described DARTS solvers for supporting the dynamics simulation of robot mechanisms. Physics-based simulations build upon such dynamics solvers and typically include several other models for engineering platform simulations. Such component models can include models for actuators and sensors, control elements, and environment models. These models interact with the dynamics as generators of forces and consumers of data. It is not unusual for such component models and their parameters to number in the hundreds and thousands for even moderately sized robotic platforms. Managing the data flow and interconnects across them can be a complex and error prone endeavor.

We use the *Darts Shell (Dshell)* simulation framework [19] for developing full-scale engineering scale simulations of robotic platforms. The goals of the Dshell framework are to facilitate the development and reuse of component models; provide standard simulation facilities such as logging, introspection, checkpointing, user interfaces; help hierarchically organize and reuse sub-system models; and meet the simulation needs for multiple domains. Figure 5 illustrates examples of systems simulated using Dshell. The adaptation of Dshell for mobile robot simulation is the *ROAMS* rover simulator [20], and for flight dynamics simulations is the *DSENDS* simulator [21]. ROAMS and DSENDS differ in the types of models they use, their specific user communities and the user interface and usage. However they are both built upon the Dshell layer, and thus able to share and use component models developed for either of the domains.

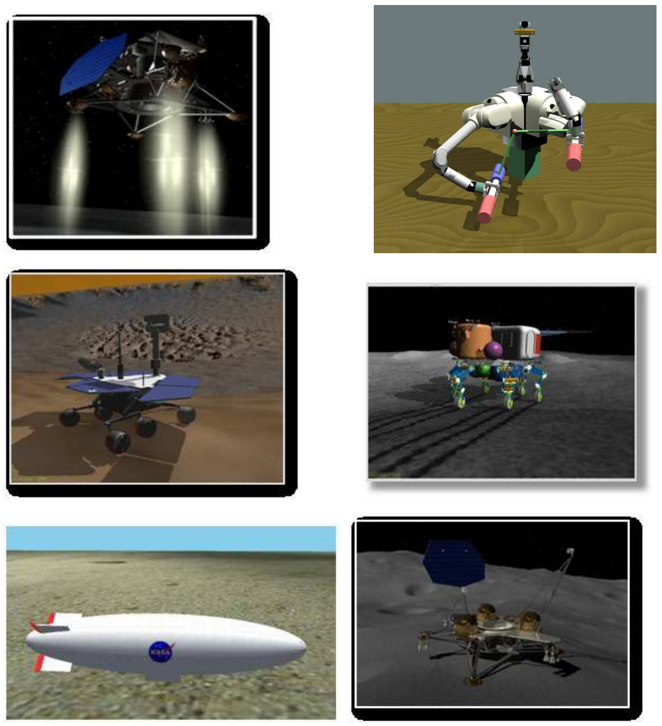
Fig. 5. Examples of platform simulations developed using the Dshell simulation framework.

The Dshell model base class supports parameters and internal states. The inputs and outputs of Dshell models can be connected at run-time to set up a data flow. The models are DARTS aware so that they can get and set dynamics data. Dshell contains an instance of the DARTS solver for propagating the dynamics and model continuous states. Bridging the gap between the low level Dshell models and the system level simulation are a hierarchy of Dshell assemblies. These assemblies represent sub-systems (eg. a wheel assembly), multiple instances of whom can be contained within assembly hierarchies. An example of the assembly hierarchy for a ROAMS rover model is shown in Figure 6. Once again, while Dshell is primarily C++ software, its Python bindings permit easy configuration, customization and scripting of the simulations.

VI. APPLICATION: EMBEDDED MODELS

The mechanical, geometric, kinematic, dynamics, sensor and actuator characteristics of the robot fundamentally effect the operation of the robot and its interactions with the task environment. Onboard autonomy modules often rely on models to obtain information and data needed during run-time. The effective use of embedded models does not require

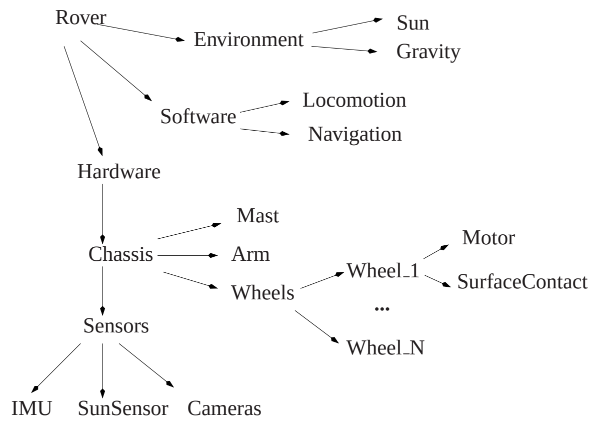

Fig. 6. Illustration of a hierarchy of Dshell assemblies used to organize the sub-systems within a ROAMS planetary rover simulation.

the models be perfect, but instead that they help reduce the demands on the autonomy software. A-priori knowledge and online estimators are typically used to continuously update and improve the model data to handle uncertainty and changes during task execution.

Due to the variety and complexities of modeling needs, there is a tendency towards using simplified or fragmented implementations of custom models to meet specific autonomy module needs. This not only adversely impacts the quality of the models but of the overall autonomy architecture. To avoid this, we have used DARTS to develop the RoboDarts embedding modeling layer for use within robot autonomy software. RoboDarts serves as an oracle-like modeling layer for embedded use within autonomy modules. It is designed to support fast computation of the broad variety of model-based information needed by the modules. Furthermore, SOA's structure-based algorithms are able to adapt quite naturally to handle run-time changes to the robot, its tasks and the environment.

RoboDarts allows the definition of different parts of the robotic systems such as arms, legs, necks, grippers etc. Specialized *forward* and *inverse kinematics* functions for any of these sub-systems can be defined. The frames layers are used by all modules for computing needed pose transforms as well as for driving 3D visualization graphics. The inverse kinematics is used for motion planning, as well as for redundancy management during real-time control. *Jacobians* and *manipulability measures* are available for all linkages in the system. These are used for planning, motion control as well as for projecting end-effector forces into joint torques.

A *collision detection* module (built upon the Bullet collision detection library [4]) supports the checking of collisions between bodies. This is used by the motion planner to plan collision free paths as well as by the grasp planner to generate grasp sets. This module allows one to select the coarseness of the collision shape geometries, to selectively hide objects, as well as to add padding to shapes. The selective collision filtering feature allows users to disable collision checking between specific pairs of bodies as needed. Collision filters are automatically updated when bodies are attached and detached from each other during run-time (eg. impact driver and its battery). This also extends to the run-time grasping and ungrasping of objects by the arm hands. Such grasping is not limited to individual rigid body objects but also to *articulated* task object linkages (eg. trimmers). Pose estimates generated by the estimator are used to continually update the attachment poses within the modeling layer.

When heavy objects (eg. tires, impact drivers) are grasped by a hand, appropriate gravity compensation torques need to be applied to overcome arm sag. Such *feedforward compensating torques* are computed by the modeling layer for the control module. Dual-arm manipulation introduces loop constraints within the system topology. While reducing the available number of motion degrees of freedom, these constraints lead to internal forces that build up within the robot. A move/squeeze decomposition approach [6] is used to compute the feedforward terms that optimally *load balance* the torques among the arms.

Dynamics simulations are also available for use by the task planner to plan execution sequences, and by the estimator to generate dynamics based predicts. Other miscellaneous computations available within RoboDarts include those for the *operational space inertia* for control in task space, and recursive methods to compute system center of mass and composite rigid body inertias.

An adaptation of RoboDarts is currently in use by the JPL team participating in the DARPA ARM-S competitive program [1]. The goals of the ARM-S project are to demonstrate autonomous dual-arm manipulation for a variety of tasks in an unstructured environment. Examples of tasks include using a key to open a door, hang up a phone, drill a hole, change a tire, cutting wire with a tool etc. The RoboDarts modeling layer is used by each of the perception, planning, estimation and control autonomy modules, though each module uses its instance of RoboDarts in a different way. Such sharing has several benefits including - centralization of model data, common model interface throughout the system, ease of updating modeling methods, and encapsulation of model complexity within the modeling layer.

VII. CONCLUSIONS

This paper describes a computational architecture that comprehensively addresses a broad range of robotics modeling needs across analysis, simulation and embedded modeling applications. The computations use fast, structure-based algorithms from the SOA theoretical framework. The richness of the architecture avoids fragmented and over-simplified model implementations. Also, the grounding in the SOA framework and the generic DARTS library allows for a continual advancement of the architecture with new capabilities and algorithms and improvement in the application layers. We believe that such strong foundations are essential for meeting the ever growing needs for autonomous robotics systems.

ACKNOWLEDGMENTS

The research described in this paper was performed at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under a contract with the National Aeronautics and Space Administration under a contract with the National Aeronautics and Space Administration and funded through the internal Research and Technology Development program.[1] This project was also supported in part by the DARPA Autonomous Robotic Manipulation Software Track (ARM-S) program.

REFERENCES

[1] N. Hudson, T. Howard, J. Ma, A. Jain, M. Bajracharya, S. Myint, C. Kuo, L. Matthies, P. G. Backes, P. Hebert, T. Fuchs, and J. W. Burdick, "End-to-End Dexterous Manipulation with Deliberate Interactive Estimation," in IEEE International Conference on Robotics and Automation, Minneapolis, MN, 2012.

[2] "Gazebo." http://gazebossim.org

[3] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct. 2012, pp. 5026–5033.

[4] "Bullet Physics Library," 2013. http://bulletphysics.org

[5] "OpenRave," 2013. http://openrave.org/docs/latest_stable/

[6] A. Jain, Robot and Multibody Dynamics: Analysis and Algorithms. Springer, 2011.

[7] ——, "Graph Theoretic Foundations of Multibody Dynamics Part I: Structural Properties." Multibody System Dynamics, vol. 26, no. 3, pp. 307–333, June 2011.

[8] G. Rodriguez, A. Jain, and K. Kreutz-Delgado, "A spatial operator algebra for manipulator modeling and control," International Journal of Robotics Research, vol. 10, no. 4, p. 371, 1991.

[9] J. Y. S. Luh, M. W. Walker, and R. P. Paul, "On-line Computational Scheme for Mechanical Manipulators," ASME Journal of Dynamic Systems, Measurement, and Control, vol. 102, no. 2, pp. 69–76, June 1980.

[10] R. Featherstone, Robot Dynamics Algorithms. Kluwer Academic Publishers, 1987.

[11] M. W. Walker and D. E. Orin, "Efficient Dynamic Computer Simulation of Robotic Mechanisms," ASME Journal of Dynamic Systems, Measurement, and Control, vol. 104, no. 3, pp. 205–211, Sept. 1982.

[12] "Open Dynamics Engine." http://www.ode.org/

[13] A. Jain, "Multibody graph transformations and analysis Part II: Closed-chain constraint embedding," Nonlinear Dynamics, vol. 67, no. 3, pp. 2153–2170, Aug. 2012.

[14] A. Jain, C. Crean, C. Kuo, H. von Bremen, and S. Myint, "Minimal Coordinate Formulation of Contact Dynamics in Operational Space," in Robotics Science and Systems, Sydney, Australia, 2012.

[15] A. Jain, "Minimal Coordinates Formulation of Contact Dynamics," in Multibody Dynamics 2013, ECCOMAS Thematic Conference, Zagreb, Croatia, 2013.

[16] A. Jain, C. Crean, C. Kuo, and M. B. Quadrelli, "Efficient Constraint Modeling for Closed-Chain Dynamics," in The 2nd Joint International Conference on Multibody System Dynamics, Stuttgart, Germany, 2012.

[17] "Simplified Wrapper and Interface Generator (SWIG)." http://swig.org

[18] A. Jain and J. Ziegler, "PyCraft: An Algorithm Workbench for Computational Multibody Dynamics," in 7th World Congress on Computational Mechanics, Los Angeles, CA, July 2006.

[19] C. Lim and A. Jain, "Dshell++: A Component Based, Reusable Space System Simulation Framework," in Third International Conference on Space Mission Challenges for Information Technology (SMC-IT 2009), Pasadena, CA, July 2009.

[20] A. Jain, J. Balaram, J. Cameron, J. Guineau, C. Lim, M. Pomerantz, and G. Sohl, "Recent Developments in the ROAMS Planetary Rover Simulation Environment," in IEEE 2004 Aerospace Conf., Big Sky, Montana, 2004.

[21] J. Cameron, J. Balaram, A. Jain, C. Kuo, C. Lim, and S. Myint, "Next Generation Simulation Framework for Robotic and Human Space Missions," in 2012 AIAA Space, Pasadena, CA, 2012.

[1] ©2013 California Institute of Technology. Government sponsorship acknowledged.