INTERNATIONAL DESIGN CONFERENCE - DESIGN 2010
Dubrovnik - Croatia, May 17 - 20, 2010.

DESIGN 2010

PROPOSAL FOR A GUIDELINE TO INTEGRATE KINEMATICS WITHIN LIGHTWEIGHT FORMATS

M. Eigner, F. J. Gerhardt, T. Gilz and and S. Handschuh

Keywords: product lifecycle management, simulation, neutral data formats, JT, kinematics

1. Introduction

Global markets and distributed engineering networks require cost-efficient and effective methods to provide product data for multiple divisions and stakeholders. Today, processes often rely on native Computer Aided Design (CAD) files. Recently, however, a change in paradigm towards integrating lightweight data formats can be monitored. In particular JT (Jupiter Tessellation) is a highly accepted format within various industries. The successful triggering of the JT ISO (International Organization for Standardization) standardization is now further driving JT establishment within software products related to Product Lifecycle Management (PLM). Additionally, two internationally renowned associations (ProSTEP iViP association and Verband der Automobilindustrie) have been engaged in defining requirements towards industrial JT application in the form of Use Cases. In the scope of underlying project activities, we have been responsible for major contributions to this work.
Based on documents created within these activities and previous investigations (see [Gerhardt 2009] [Gerhardt Eigner, 2009]), conclusions can be drawn regarding the support of downstream processes, such as Digital Mock-Up (DMU) and Simulation. Even though kinematical information plays a key role within product validation, we know of no lightweight (CAD-derived) format that can store such data. Accordingly, support for respective Use Cases is not directly given. As of today, formats like JT do not provide containers to store joints, fixes and links with relevant attributes, all of which are used e.g. in multibody-simulation (MBS) and installation feasibility analysis.
Today, kinematical simulations are primarily run within systems specifically developed for this purpose, relying on respectively native data formats. CAD is then linked to such Computer Aided Engineering (CAE) systems via often expensive tool-specific interfaces. Our goal is to serve processes with kinematical data via lightweight formats. We propose a guideline that maps such information in the product structure. In addition to storing kinematical connections within such data formats, we have mapped those contents to a well established MBS-format called Physical Modelling XML as an intermediate solution, again bridging the gap between CAD and CAE.
The remainder of this paper is structured as follows: Section 2 gives an overview on state of the art regarding kinematical representations in different data models. Section 3 then provides insight into similar publications, before Section 4 presents the mentioned approach and proposal for a guideline. Results of our work in terms of a prototypical implementation are illustrated in Section 4.2, before Section 5 wraps up with an outlook on work and trends to come from our point of view.

2. State of the art

A rigid Multibody-system is primarily described by two aspects: kinematics and the forces that actually cause motion in the system. We restrict our approach to kinematics only, meaning the interconnection of bodies. Before addressing a guideline to map kinematics within lightweight data formats, two topics were considered:

1. How kinematical data is typically prepared in the context of CAD
2. Data formats and kinematical descriptions existent today

Regarding the first issue, two approaches can be distinguished:

Approach	Description
Geometrical constraints:	For a Multibody-system consisting of n bodies, 6n degrees of freedom (DoF) are assumed, a body having six DoF: 3 translational and 3 rotational. Adding constraints that relate to geometrical elements in the CAD-system, e.g. “point moves on curve”, reduces the number of DoF.
CAD-based joints:	Some CAD-systems, specifically more comprehensive ones (e.g. CATIA and NX), provide modules to specify joints, representing an interconnection between two bodies. As above, these constrain the number of DoF in the system, which is why some of the descriptions dealing with kinematics consider joints to be a subset of constraints in general.

Since our main focus lies on CAD-based joint definitions, we consider respective CAD functionality a prerequisite. Regarding the second issue, we have studied four different data formats, in part much rather description languages, prior to considering the guideline-approach presented in Section 4. From here on we shall call them “descriptions”, each positioning itsef in a slightly different key influential area. In the following, each is elaborated in more detail.

2.1 PLM XML

PLM XML is a comprehensive data format based on XML and is considered an industrial standard. It has historically evolved from facilitating product lifecycle interoperability [Siemens 2009] in the context of applying several Mechanical CAD-(MCAD) systems in cross-enterprise engineering. Support for managed, revision- and view-based product data as used in common PDM-systems, is included.

PLM XML stores product data in a compact instance graph, allowing concretion via product views based on occurrences that represent a more natural tree-like hierarchy. The same approach is followed for the definition of a kinematical system (mechanism), with the exception that occurrences which can be linked to occurrences in a product view are placed directly into so called *MechanismRevisionView* (MRV) graph elements that represent either links or sub-mechanisms, and that there exist no views on a mechanism. For details, we advise studying the PLM XML schema that can be found online.

PDM is a key topic influencing PLM XML contents, including kinematical definitions. On one hand, this provides flexibility, while on the other we consider it to be little intuitive. Regarding the key elements of a kinematical definition, namely joints, PLM XML understands these as a certain form of constraint. Constraints are at first placed independent of the mechanism, but any MRV can then reference instances of a constraint via a respective attribute. Constraint instances in turn typically reference so called Marker elements, each representing a point (with orientation) contained in the MRV definitions.

Joints are classified into primitive and other joints, both restricting motion between two markers (each on a different link) to one or more DoF. A primitive joint is characterized by restricting either translational or rotational degrees, but never both. Neither does a primitive joint “mechanically couple” the links, making this type of joint uninteresting for our multibody considerations. Other joints can restrict alike or more complex, but there exists no direct correlation between primitive and other joints. This is different in other descriptions, such as Physical Modelling XML for example.

2.2 Physical Modelling XML

Physical Modelling XML (PhysMod) is based on SimMechanics^{TM}, a single-domain extension of the Simulink environment developed by The Mathworks, Inc, which can be used for modeling and simulation of mechanical systems. As PLM XML, it is based on standard XML.

Being a CAE-oriented data format, only information relevant for this purpose is provided. PhysMod files consist primarily of bodies, grounds and joints. Compared to PLM XML, neither product structure nor further PDM data is included. External geometry (e.g. for simulation visualization) can be referenced per body.

A body represents a component that is linked to another one. Each is accompanied by sub-elements representing the properties required for simulation, in particular mass, inertia and respective units. In analogy to PLM XML, bodies are given here called frames that have a unique reference identifier. Essentially, a frame is a coordinate system of interest (markers are the PLM XML pendant). Each body must hold at least a CG (center of gravity) and a CS1 (origin of the body) frame.

A joint links two bodies via their frames by a base-follower mechanism, removing DoFs from the system as a whole. PhysMod distinguishes between primitive (*revolute*, *prismatic* and *spherical*) and complex joints, which are a combination of the primitive ones. This is different from e.g. PLM XML. An axis element is associated with each primitive, interpreted according to the primitive's type (axis to translate on, rotate around or representing a pivot point). In the XML file, each joint is represented by a sequential list of "Primitive" elements, motions to then be interpreted in order of appearance.

A ground can be interpreted as a body with infinite mass that can be used as a fixed reference point for other elements of the XML file (e.g. frames positioned relative). Each kinematical model must at least store one ground that is joined to a body via a joint. Good practice is to create a body called RootPart and a ground called RootGround.

2.3 COLLADA

COLLADA is short for COLLAborative Design Activity and defines an XML-schema to enable 3D authoring applications to freely exchange digital assets, including graphics, animation, kinematics, physics and shader effects without loss of information [Barnes, Levy, 2008].

While COLLADA has grown, today serving as the base technology e.g. within the AutomationML group [Drath, Lüder, Peschke, 2008], it primarily stores data of a visual scene in a scene graph, pointing out its key origin: the gaming industry and computer animation. There can be multiple physical (COLLADA 1.4 Physics) or kinematical scenes (COLLADA 1.5 Kinematics), describing with rigid constraints, how one body can move in relation to another. We consider the second.

Just like (in game engines) 3D-objects are commonly stored within model definitions and instanced in a visual scene, COLLADA can store kinematical models that are later instanced and linked to geometry representations in the scenegraph. A kinematic model, in turn instancing separately defined joints, is defined by kinematic chains that consist of links and joints. Links represent rigid kinematical objects without mass and inertia, whose motion is constrained by one or more joints using a so called "attachment" mechanism. An important issue is the fact that COLLADA provides only the primitive "prismatic" and "revolute" joints, both consisting of a single axis (and limits); all other joints are considered a combination of the above. This is similar to Physical Modelling XML, with the exception that a spherical joint is no primitive. These axes are linked to axes defined in the visual scene.

2.4 Modelica

Modelica is more an object-oriented language than a format. Supporting declarative multi-domain modeling, our focus lies on modeling kinematics. For this purpose, a standardized and free *MultiBody* Library is provided, holding 3D mechanical components to model mechanical systems conveniently. Every model utilizing the *MultiBody* library must have an instance of *World*, defining a gravitational field and global coordinate system fixed in 3D space. Specializing *Part* (containing mass, inertia, position and at least one so called *frame*, which is basically a coordinate system), rigid bodies within the kinematical definition are represented by *Body*, the most interesting being the general *BodyShape*. Such bodies possess two frames, *frame_a* and *frame_b*, which are used for interconnection e.g. by a joint. A joint too possesses two frames, allowing parts to be connected on both "ends". *Fixed* is another "part" of interest, representing a fixed position in the world [Fritzson, 2004], in analogy to the Ground container in Physical Modelling XML.

3. Related work

A first step towards modeling kinematics within lightweight data formats was to develop an application that lets the user view product structure, geometry and further contents, such as node properties. This was done in previous work. The architecture and functionality of that application was first presented and elaborated in [Gerhardt, Eigner, 2009], working specifically on the JT data format.
Against the background of working with kinematics, there have been rather recent works dealing with CAD to CAE connections. Juhász et al. have presented partially automated workflows to translate from specifically Pro/Engineer CAD models to Modelica, by using Physical Modelling XML as an in-between format [Juhász, Schmucker, 2009]. Also working with the Modelica language, Engelson et al. have implemented a translator to create a Modelica model from an existing AutoDesk's Mechanical Desktop CAD model [Engelson, et al., 2003]. Welakwe et al. on the other hand have developed an interface that integrates Modelica into CATIA, enabling automatic extraction of pertinent information from the CAD representation of physical components to Modelica models for multibody simulation [Bhattacharya, et al., 2006].
All of the above references relate to integrating a well-known description language, namely Modelica, with a given CAD-system. While these works provide very promising results, none address the importance of going from CAD to a lightweight data format, and then to CAE. We consider this essential, aiming for the integration of such formats into the supply chain. We are not aware of any other works dealing with this matter in the scope of kinematical representations.

4. Approach "abusing" the product structure for kinematics

Without loss of generality, the approach is presented based on the JT data format. Since basically all of the lightweight formats we have come across (JT, 3D XML, PRC and ProductView) provide means to store product structure, the concept remains transferable. Section 4.1 presents the approach in general; Section 4.2 illustrates an implementation thereof using our developed application (cp. Section 3).

4.1 The approach to enhance the product structure

For all of the above descriptions, some pre-definition or classification of joints is provided. A classification into primitive and complex (or compound) joints that consist of primitives was considered straight forward and adapted within the persented approach.

Table 1. Joint types

Joint name		**PhysMod pendant**	**PLM XML pendant**	**Modelica pendant**	**COLLADA pendant**
Primitive					
Revolute	**(R)**	Revolute	Revolute	Revolute	Revolute
Translational	**(T)**	Prismatic	Translational	Prismatic	Prismatic
Complex					
Spherical	**(RRR)**	Spherical	Spherical	Spherical	Compound
PlanarNoRevolute	**(TT)**	Planar	N/A	N/A	Compound
PlanarRevolute	**(TTR)**	In Plane	Planar	Planar	Compound
Cylindrical	**(TR)**	Cylindrical	Cylindrical	Cylindrical	Compound
Universal	**(RR)**	Revolute-Revolute	Universal	Universal	Compound

Joints were identified as depicted in Table 1 relating to COLLADA for the primitives (providing a single DoF for one body in relation to another) and also already defining a set of complex ones, primarily similar to Physical Modelling XML. The key joints illustrated in Table 1 were integrated into the approach. The specification of a set of complex joints provides means to coarsely model early kinematics. A universal joint, for example, can be modeled precisely via primitive joints by providing a body that represents the cross-shaped linkage of the two connected shafts. Defining a revolute joint between the first shaft and linkage, and one between the linkage and the second shaft, leads to the intended behavior of the model. In case of missing linkage geometry (e.g. in early phases of product development) or a less exact definition of the behavior, the availability of a universal joint allows the definition of two rotational axes within one joint, requiring only shaft bodies. This is illustrated in Figure 1.

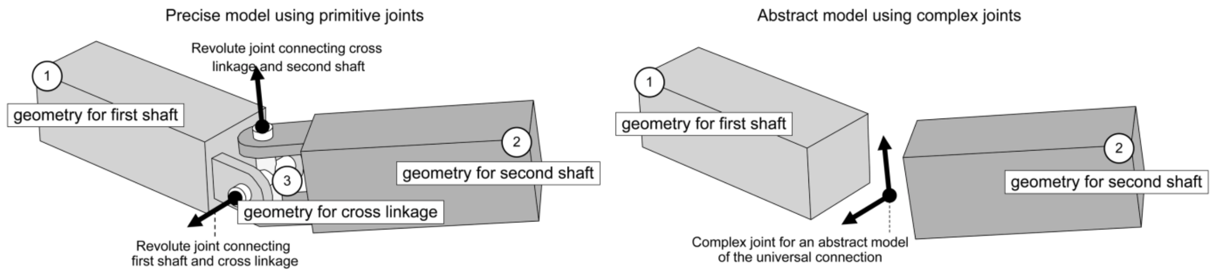
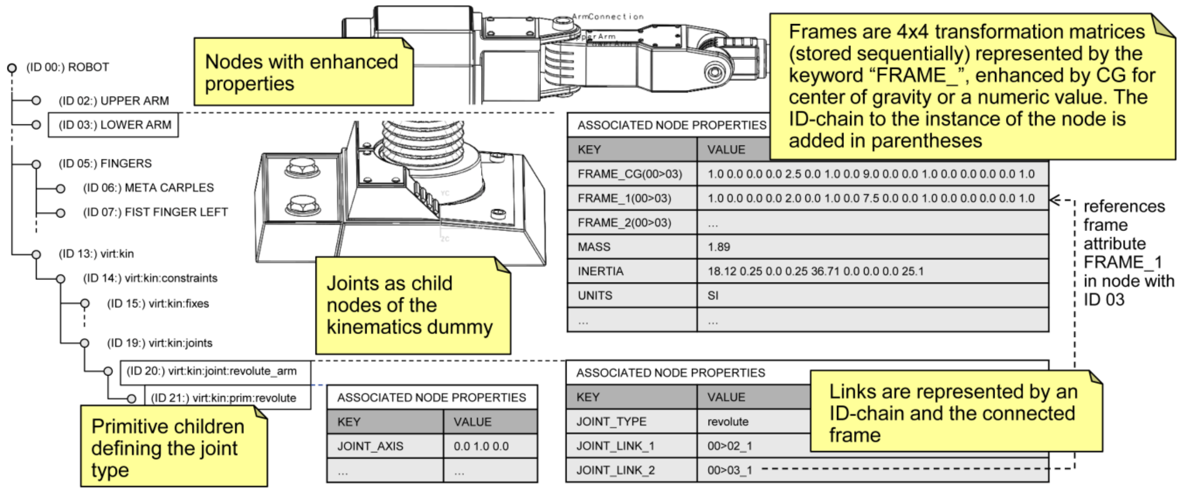

Precise model using primitive joints
Revolute joint connecting cross linkage and second shaft
1
geometry for first shaft
2
geometry for second shaft
3
geometry for cross linkage
Revolute joint connecting first shaft and cross linkage
Abstract model using complex joints
1
geometry for first shaft
2
geometry for second shaft
Complex joint for an abstract model of the universal connection

Figure 1. Primitive vs. complex joints

Because we neglect forces that produce the motion, actuated joints that are an inherent part of e.g. Modelica are not regarded. We consider such information to be subject to downstream detailing and instancing of a simulation. For each primitive, the following joint attribute is essential:

Revolute:	axis of rotation (a vector with orientation)
Translational:	axis of translation (a vector with orientation)

A product structure, consisting of assembly and part nodes, could at any time be enhanced by what we call ‘dummy nodes’. These don’t represent actual components of a product, but much rather virtual nodes that further describe the product. We propose enhancing the product structure via a kinematical dummy that indicates that all attached sub-nodes now contain kinematical content. Figure 2 depicts the general approach based on a simplified model, here displaying a single primitive revolute joint.


(ID 00:) ROBOT
(ID 02:) UPPER ARM
(ID 03:) LOWER ARM
(ID 05:) FINGERS
(ID 06:) META CARPLES
(ID 07:) FIST FINGER LEFT
(ID 13:) virt:kin
(ID 14:) virt:kin:constraints
(ID 15:) virt:kin:fixes
(ID 19:) virt:kin:joints
(ID 20:) virt:kin:joint:revolute_arm
(ID 21:) virt:kin:prim:revolute
Nodes with enhanced properties
Joints as child nodes of the kinematics dummy
Primitive children defining the joint type
Frames are 4x4 transformation matrices (stored sequentially) represented by the keyword “FRAME_”, enhanced by CG for center of gravity or a numeric value. The ID-chain to the instance of the node is added in parentheses
references frame attribute FRAME_1 in node with ID 03
Links are represented by an ID-chain and the connected frame
ArmConnection
UpperArm

ASSOCIATED NODE PROPERTIES

KEY	VALUE
FRAME_CG(00>03)	1.0 0.0 0.0 0.0 2.5 0.0 1.0 0.0 9.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0
FRAME_1(00>03)	1.0 0.0 0.0 0.0 2.0 0.0 1.0 0.0 7.5 0.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0
FRAME_2(00>03)	...
MASS	1.89
INERTIA	18.12 0.25 0.0 0.25 36.71 0.0 0.0 0.0 25.1
UNITS	SI
...	...

ASSOCIATED NODE PROPERTIES

KEY	VALUE
JOINT_AXIS	0.0 1.0 0.0
...	...

ASSOCIATED NODE PROPERTIES

KEY	VALUE
JOINT_TYPE	revolute
JOINT_LINK_1	00>02_1
JOINT_LINK_2	00>03_1

Figure 2. Sketch of the approach in general

On a general note, we propose a common understanding regarding terminology that identifies a node as virtual, no matter the type of dummy. This way, any processor or application can filter such nodes with respect to arbitrary purposes, such as hiding them from display in a product structure tree. Consistent with its meaning, we suggest virtual nodes to be prefixed ‘virt:’. Hence, the kinematical dummy node is called *virt:kin*, which is short for virtual:kinematics. This node is attached a child assembly called *virt:kin:constraints*, in turn attaching a *virt:kin:joints* and a *virt:kin:fixes* child assembly. Now, a joint is defined by attaching a child assembly called *virt:kin:joint:T_N*, where

- T is a placeholder for the type of joint (name as specified in Table 1)
- N is an arbitrary name for the joint

The remainder of the joint is specified via key (string of characters) and value paired properties, as shown in Table 2, and a list of child parts *virt:kin:joint:prim:P* representing its primitives, where

- P is a placeholder for the type of primitive (name as specified in Table 1)

In case the joint is not complex, only one primitive child is provided.

Table 2. Joint properties

Property key	Property value (description)
JOINT_TYPE	String of characters defining the type of joint conform to Table 1
JOINT_LINK_1	String of characters representing a reference to a frame within the product node that defines the base (recall directionality of a joint): I_F, where I represents the node's ID chain and F the frame (CG or a number)
JOINT_LINK_2	String of characters representing a reference to a frame within the product node that defines the follower (recall directionality of a joint): I_F, where I represents the node's ID chain and F the frame (CG or a number)

Table 2 mentions the storage of a node's identifier (ID)-chain within the link definition. Further detailed in Definition 1, this specific form of notation represents a mechanism to uniquely reference nodes. I_1 within an ID chain is most commonly the root node of the considered data set. The node reference via an ID chain also applies to links in a fix, as specified in Table 3.

Definition 1: *The **ID chain** of a node n within a product structure is a sequence of node identifiers "I_1>I_2>...>I_n", for which the following conditions hold:*

- I_n is the identifier of the node n itself.
- I_i is the identifier of the i-th node in that path of the product structure, which by traversal, beginning at the node with identifier I_1, leads to n.

The "META CARPLES" node in Figure 2 would hence be referred to using the substring of characters 0>3>6. The reason for this form of notation is that a data format may provide means to store instances of parts and assemblies. While an instance node itself has an individual ID in the product structure, the sub-tree it references doesn't. This leads to different parts or assemblies sharing the same ID, as exemplary illustrated in Figure 3.

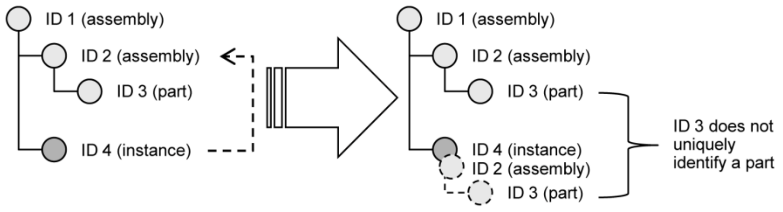


Figure 3. Instances leading to non-unique part identifiers

Primitive nodes are again defined via a single key and value paired property as shown in Table 3.

Table 3. Primitive properties (top) and Fix properties (bottom)

Property key	Property value (description)
JOINT_AXIS	String of characters "X_1 X_2 X_3", defining either the rotational or translational axis, depending on the type of primitive (as described above). X_i represent the coordinates.
Property key	**Property value (description)**
FIX_LINK_1	String of characters representing a reference to a frame within the product node that defines the base: I_F, where I represents the node'ss ID chain and F the frame (CG or a number). **This string can be empty, representing that the second link is fixed to the**

	“ground” (virtual body of infinite mass) of the kinematic system.
FIX_LINK_2	String of characters representing a reference to a frame within the product node that defines the follower: I_F, where I represents the node’s ID chain and F the frame (CG or a number).

Fixes are specified analog to joints, except that there is no need to hold child nodes that represent primitives. There must also be a common sense regarding body properties that are relevant for a kinematical model. This includes frames that are referenced by the joints, as well as the mass and inertia of a body. These properties are shown in Table 4. While “MASS”, “MOMENT_OF_INERTIA”, “FRAME_CG” and “FRAME_1” are considered obligatory, the other m-2 frames (“FRAME_2” to “FRAME_m”) represent optional properties.

Table 4. Body properties

Property key	Property value (description)
MASS	A floating point value
UNITS	String of characters “SI[P]”, where [P] is optional, representing one of the following prefixes: “milli”, “centi” or “deci”. This property refers to the units that underly the subsequent properties.
MOMENT_OF_ INERTIA	String of characters “I_{xx} I_{xy} I_{xz} I_{yx} I_{yy} I_{yz} I_{zx} I_{zy} I_{zz}”. Hereby, I_{xy} (= I_{yx}), I_{xz} (= I_{zy}) and I_{yz} (= Izy) represent the centroidal products and I_{xx}, I_{yy} and I_{zz} the centroidal moments.
FRAME_CG (I_1>I_2>...I_n)[D]	String of characters “X_0 ... X_{15}” holding the node’s row-major *center of gravity* coordinate system transformation, in relation to the world coordinate system. [D] is optional, D being a string of characters describing the frame.
FRAME_1 (I_1>I_2>...I_n)[D]	String of characters “X_0 ... X_{15}” holding the node’s row-major *origin* coordinate system transformation, in relation to the world coordinate system. [D] is optional, D being a string of characters describing the frame.
FRAME_2 (I_1>I_2>...I_n)[D]	String of characters “X_0 ... X_{15}” holding an *arbitrary* row-major coordinate system transformation, in relation to the world coordinate system. [D] is optional, D being a string of characters describing the frame.
…	…
FRAME_m (I_1>I_2>...I_n)[D]	String of characters “X_0 ... X_{15}” holding an *arbitrary* row-major coordinate system transformation, in relation to the world coordinate system. [D] is optional, D being a string of characters describing the frame.

As can be seen, the notation for the frame keys again makes use of ID chains (compare fix and joint links). The path-based form of notation allows the definition and later retrieval of frames for selective instances of the node. A node can store an arbitrary number of all frames, given different ID chains.
Summarizing, the approach relies on “abusing” the product structure by adding virtual nodes in combination with respective properties. We manage to integrate kinematics within data formats whose key competence lies in efficient visualization. Using the guideline proposed above, we have enhanced the application developed in [Gerhardt Eigner 2009] to prepare a JT file by adding kinematical content. A JT can then be exported into a PhysMod file to generate SimMechanics^{TM} building-blocks for simulation. Tasks associated with preparing the bridge from JT to CAE are elaborated in the following.

4.2 Prototypical implementation

For the prototypical definition of kinematical content within a JT-dataset, a minimal workflow-based technology relying on CAD-reference geometry was chosen. While some of the content required for kinematical models is not provided by default exports from CAD to JT, the implementation of new and specific CAD to JT interfaces was neglected. An alternative solution to supply the required information was provided, sufficing the prototype’s main purpose to serve as proof of concept. Based on the design decisions made, prototypical preparation and validation of kinematical contents in a JT file has involved four steps, as elaborated subsequently.

4.2.1 Export from CAD to JT

As of today, there exists no solution to exporting kinematics from CAD to JT. Such data must be specified in our neutral application. In analogy to defining kinematics inside a CAD-system, we have chosen to use auxiliary geometry to do so (reference planes, axes and points). For the JT to carry reference geometry into our application, it is to be assured that the option to include such information is set in the JT translator export settings, or by manually manipulating the JT configuration file that most translators rely on. By the example of the Siemens PLM NX6 JT translator, the configuration file requires the entry activateCsysPMI = true.

4.2.2 Enriching JT with further CAD-derived properties

Bodies within the created JT need to be enriched with the properties required for the model to be used for simulation. In particular, this includes mass, inertia and center of gravity as described previously. While such properties can be calculated in all major CAD-systems on the market, there exists no option in the JT-configuration-file to include these. Until this is solved, means to load such data into the application retroactively as a temporary solution were provided. Most CAD-systems provide ways to store ASCII-based descriptions of the required properties calculated per part, amongst others using the material information included in the part's definition. A class called *CCADReader* was developed and implemented. Given the path to an ASCII-file and a string of characters describing the CAD-system it was created from, the reader parses the file and stores the relevant information for later retrieval. So once a part is selected and the user triggers an enrichment through navigation in the application's main window, he is asked to specify file name and CAD-system via a respective dialog, which is then passed to the main controller of the prototype. The parsing procedure is started. Once finished, the controller dynamically updates the selected part's properties. In addition to the center of gravity frame, the prototype creates the FRAME_1(...) property upon enriching a part. For this purpose, OpenSceneGraph provides internal functionality to extract the world matrix of a specific node. In future, the mentioned properties should be auto-generated by the export tool creating the JT (there should at least be an option to do so). Once parts are adequately prepared, further frames, joints and fixes can be defined. Adding frames is not further elaborated in this paper.

4.2.3 Defining joints and fixes

A minimal workflow technology to guide the user through the definition of frames as described in Table 4, kinematic joints and fixes was implemented. A workflow is triggered via push-buttons inside a "Multibody Modeller" dock widget. Each workflow consists of workflow items, allowing part geometry or a frame to be associated. The way kinematic data preparation is met is best explained by an example. Considering the user wants to add a primitive revolute joint, the underlying workflow's items are traversed sequentially: the user first selects a body (geometry that belongs to a specific JT node). A pointer to the body is stored in the workflow item. The second workflow item launches a dialog, allowing the user to select and "store" a frame that is associated with the node selected in the previous workflow item, using a combo box. The same sequence is followed a second time, letting the user select body and frame to be linked in relation to one selected first. The last workflow item asks the user to select a reference axis, representing the axis of rotation.

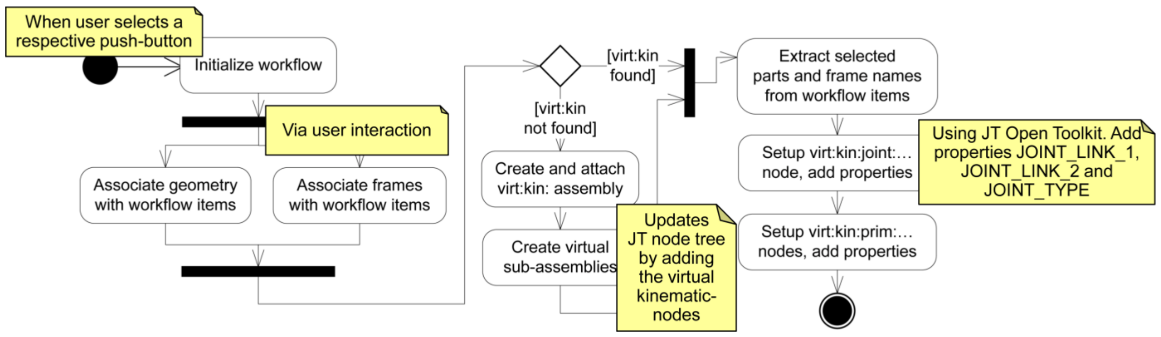

When user selects a respective push-button
Initialize workflow
Via user interaction
Associate geometry with workflow items
Associate frames with workflow items
[virt:kin found]
[virt:kin not found]
Create and attach virt:kin: assembly
Create virtual sub-assemblies
Updates JT node tree by adding the virtual kinematic-nodes
Extract selected parts and frame names from workflow items
Setup virt:kin:joint:... node, add properties
Using JT Open Toolkit. Add properties JOINT_LINK_1, JOINT_LINK_2 and JOINT_TYPE
Setup virt:kin:prim:... nodes, add properties

Figure 4. Control flow for adding a joint

Once the workflow-instance is successfully traversed, the controller first checks if the virtual node that was described in Subsection 4.1 is existent. If not, the structure is enhanced, adding that dummy and respective sub-assemblies for fixes and joints. For this purpose, JT Open Toolkit functionality is utilized. Finalizing the example, *virt:kin:joint:revolute_...* and a *virt:kin:prim:revolute* nodes are created. By extracting the selected frame names, the JOINT_LINK properties are created and added to the newly created joint node. The selected axis is used to create the JOINT_AXIS property which is added to the newly created primitive node. Generalizing independent of the joint's type, Figure 4 illustrates the control flow. Adding of a fixings is handled analogeously.

4.2.4 Physical Modelling XML export

Avoiding the development of a direct JT input interface for an MBS-tool, an objectoriented data model containing the key contents of Physical Modelling XML was created. Today, there exists no publicly available Physical Modelling XML data format reference, which is why XML-file instances in version 1.0.4 have been used to derive the data model. Utilizing a kinematics-enriched JT-file, an instance of this data model can be created. The setting up of a Document Object Model (DOM)-tree allows a simple autogeneration of a physical XML file using the XERCES C++ libraries. The DOM is created piece by piece. The resulting files can be imported into the calculation environment provided by The Mathworks Inc., using the SimMechanics^{TM} toolbox. Upon triggering an import call, the toolbox automatically creates a block diagram, whose contents can be altered if desired and launched for simulation. So called scope blocks can be added to bodies or joints for analysis of attribute courses. SimMechanics^{TM} doesn't directly support JT for visualization. For this reason, an STL-writer has been implemented, creating an ASCII ".stl" file for each body included in the Physical Modelling XML.

5. Conclusion and outlook

Figure 5 shows a proof of concept based on our approach, illustrating a block-diagram generated from a JT-dataset, and the result of a simulation that was run for 8 seconds, without any added actuators. The displayed diagram shows the course of a forearm's frame position in terms of x, y and z coordinate translation.

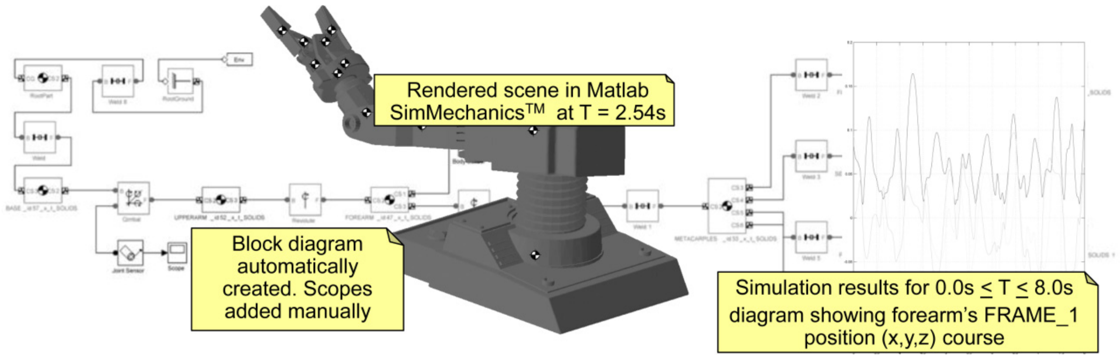

Rendered scene in Matlab SimMechanics™ at T = 2.54s
Block diagram automatically created. Scopes added manually
Simulation results for 0.0s ≤ T ≤ 8.0s diagram showing forearm's FRAME_1 position (x,y,z) course

Figure 5. Matlab simulation model created from a JT-dataset

The concept and prototypical solution that has proven the possibility to integrate kinematical connections into JT (in a manner that is adaptive to other lightweight formats), has improvement-potentials:

- Even though the definition of forces and drivers that cause the motion in a kinematical system are considered part of actually defining simulation-instances, e.g. in an MBS-tool, such information is also conceivable in JT if previously defined. As a sibling to the constraints, the concept could be enhanced by a further virtual assembly called *virt:kin:forces*. Terminology would need to be restricted and specified in a manner similar to that already contained in the concept.
- Some lighweight data formats, including JT, already suggest the storage and labelling of specific properties that relate to the creating CAD-model. In fact, The JT File Format

Reference proposes, amongst others, properties called CAD_CENTER_OF_GRAVITY, CAD_MASS_UNITS and CAD_MASS. These do not coincide with the properties described in this paper's approach, for example suggesting the terminology MASS and FRAME_CG (I_1>I_2>...I_n)[D] for mass and center of gravity. This can lead to multiple occurrences of certain properties, which is unpretty but not critical. A common sense on such CAD-derived properties would be considered a major achievement, to be realized on the level of the CAD-systems themselves though, in a vendor-comprehensive effort. In this case, it would be advisable to adopt the terminology in the here presented approach.

- At this point, only SI-units are supported for the property called UNITS. Integration of further systems may be an option, e.g. the imperial system including amongst others inches for length-units.
- Less a potential, but much rather a point worthy of mention, is the fact that kinematical sub-systems are simply realized by adding a kinematical definitions on the level of the respective sub-assembly. Accordingly, a dataset that consists of multiple assemblies and sub-assemblies might include multiple kinematical nodes (Figure 6-2). Inversely, should two assemblies with existing kinematical definitions be summarized into a super-assembly (Figure 6-1), it is to be made sure that identifiers and links within the existing joint definitions are updated.

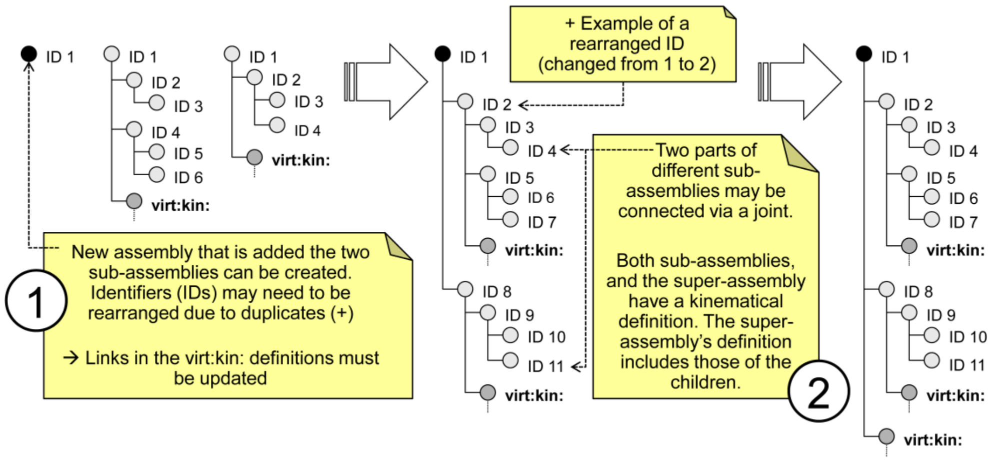

ID 1
ID 1
ID 2
ID 3
ID 4
ID 5
ID 6
virt:kin:
ID 1
ID 2
ID 3
ID 4
virt:kin:
New assembly that is added the two sub-assemblies can be created. Identifiers (IDs) may need to be rearranged due to duplicates (+)
→ Links in the virt:kin: definitions must be updated
1
+ Example of a rearranged ID (changed from 1 to 2)
ID 1
ID 2
ID 3
ID 4
ID 5
ID 6
ID 7
virt:kin:
ID 8
ID 9
ID 10
ID 11
virt:kin:
Two parts of different sub-assemblies may be connected via a joint.
Both sub-assemblies, and the super-assembly have a kinematical definition. The super-assembly's definition includes those of the children.
2
ID 1
ID 2
ID 3
ID 4
ID 5
ID 6
ID 7
virt:kin:
ID 8
ID 9
ID 10
ID 11
virt:kin:
virt:kin:

Figure 6. Kinematical sub-systems

The presentation of the guideline-proposal in more detail would have gone beyond the scope of this scientific paper. However, we believe to have illustrated the core of our work. We consider our approach to be very flexible, because it does not rely on mechanisms and containers that are specific for a data format. From our point of view, it would be most interesting to see a CAD to JT translator adapt the guideline in order to provide a more significant proof of concept. This is something we will try to go after. From a process-oriented perspective, we continue along our road to establish lightweight, neutral data formats within the product development process chain.

References

Barnes, M., Levy, E., "COLLADA – Digital Asset Schema Release 1.5.0", Specification, The Khoronos Group Inc., Sony Computer Entertainment Inc., 2008

Bhattacharya, P., Welakwe, N. S., Makanaboyina, R., Chimalakonda, A., "Integration of CATIA with Modelica", 5th International Modelica Conference, Vienna/Austria, 2006

Drath, R., Lüder, A., Peschke, J., "AutomationML – the glue for seamless Automation Engineering", ETFA 2008, IEEE International Conferece on Emerging Technologies and Factory Automation, 2008

Engelson, V., Bunus, P., Popescu, L., Fritzson, P., "Mechanical CAD with Multibody Dynamic Analysis Based on Modelica Simulation", SIMS 2003 – 44th Conference on Simulation and Modeling, Västerås/Sweden, 2003

Fritzson, P., "Principles of Object-oriented Modeling and Simulation with MODELICA 2.1", IEEE Press, 2004

Gerhardt, F., "Requirements towards establishing JT in the CAx process chain and in data exchange scenarios", ProSTEP iViP Association, ProSTEP iViP Symposium, Berlin / Germany, 2009., Track 10/2

Gerhardt, F.J., Eigner, M., "Extensible JT Open Tool for Prototypical Process Support, based on OpenSceneGraph and QT", IEEE Proceedings of the 2009 13th In-ternational Conference on Computer Supported Cooperative Work in Design, 2009, pp. 734-739

http://www.plm.automation.siemens.com, accessed 2009

Juhász, T., Schmucker, U., "Automatic Model Conversion to Modelica for Dymola-based Mechatronic Simulation", 6th International Modelica Conference, University of Applied Sciences, Bielefeld/Germany, 2009

Siemens PLM Software: "White paper - Open product lifecycle data sharing using XML",

Dipl.-Inf. Florian Jürgen Gerhardt
Research Assistant
University of Kaiserslautern, Institute for Virtual Product Engineering
Gottlieb-Daimler-Straße 44, Kaiserslautern, Germany
Telephone: +49 (0)631 2053686
Telefax: +49 (0)631 2053872
Email: gerhardt@mv.uni-kl.de
URL: http://vpe.mv.uni-kl.de