The NIST Kitting Viewer: Description and Users Manual

Thomas Kramer, Craig Schlenoff, Stephen Balakirsky, Zeid Kootbally, & Anthony Pietromartire

1 Introduction

This paper describes a software utility called the Kitting Viewer that performs a graphical simulation of a plan for building one or more kits, computes metrics about the plan, and generates a score for the plan. The Kitting Viewer is a C++ program using OpenGL graphics.

1.1 Kitting

In industrial assembly of manufactured products, kitting is often performed prior to final assembly [1]. Several different techniques are used to create kits. A kitting operation where a kit box is stationary until filled at a single kitting workstation is referred to as *batch kitting*. The Kitting Viewer deals with batch kitting processes in which a kit's component parts are staged in containers positioned in the workstation, and the locations of parts within the containers are known.

In addition to the kit's component parts, a kitting workstation usually contains a storage areas for empty kit boxes and completed kits. The Kitting Viewer deals with these.

Kitting has not yet been automated in many industries where automation may be feasible. Consequently, the cost of building kits is higher than it could be. The Kitting Viewer addresses this problem by providing a simulation environment and metrics that allow for an unbiased comparison of various approaches to building kits in an agile manufacturing environment.

The Kitting Viewer allows for variations in kit contents, kit layout, and component supply. The Kitting Viewer assumes that a robot performs a series of pick-and-place operations in order to construct a kit. These operations include:

- Pick up an empty kit tray and place it on the work table.
- Pick up multiple component parts and place them in the kit tray to make a kit.
- Pick up the completed kit and place it in the full kit storage area.

Each of these may be a compound action that includes other actions such as opening and closing a gripper and end-of-arm tool changes.

1.2 Kitting Viewer Appearance

Figure 1 shows the Kitting Viewer display. The display uses three windows, labeled Metrics & Settings, Kitting Viewer, and Kitting Command & Messages. The windows may be moved and resized independently, like other windows in a typical windowing system.

The Kitting Viewer window shows a view of the kitting workstation. The floor of the workstation is covered with a grid. The robot in the workstation is represented by a gantry robot spanning the entire width of the workstation. The gantry robot moves when any robot motion command is executed. The speed at which the picture of the robot is animated normally matches the actual commanded speed of the robot. Objects in the workstation move if the robot moves them.

Until all robot commands are executed, the Kitting Command & Messages window shows the currently executing command or the most recently executed command, if no command is currently executing. After all commands are executed, and the locations of objects in the workstation are being checked, the Kitting Command & Messages window

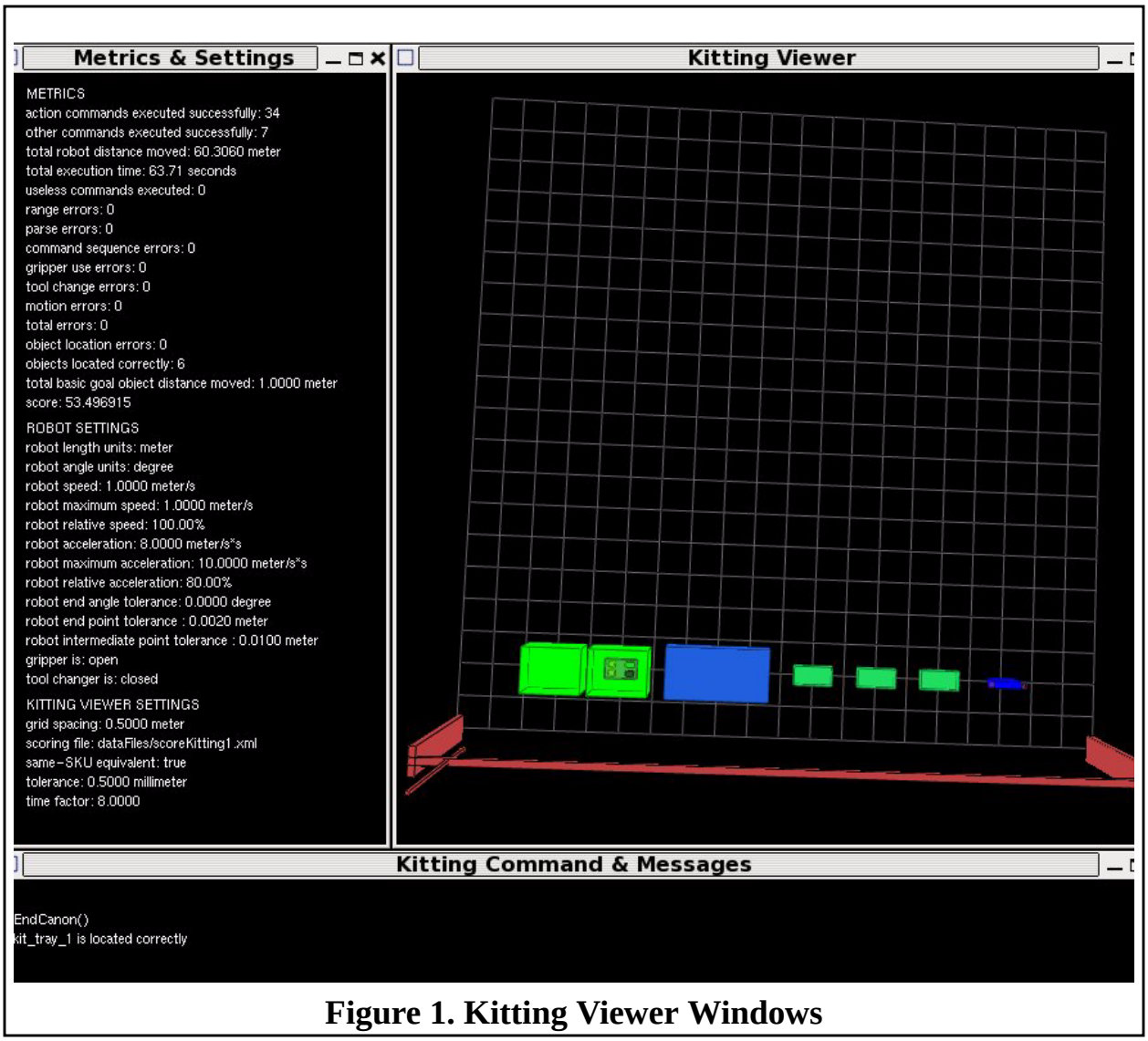

Figure 1. Kitting Viewer Windows

shows location messages. Error messages and other messages are also displayed in this window.

The Metrics & Settings window shows metrics at the top, robot settings in the middle, and Kitting Viewer settings at the bottom. The metrics are described in Section 3. All but two of the robot settings correspond to items that may be set using robot commands. The extra two are the robot's maximum speed and maximum acceleration (which may not be reset). As commands are executed, metrics and settings are updated in the window.

1.3 Kitting Viewer Functionality

The Kitting Viewer may be run with or without simulating execution of a robot program. It also lets the viewer:

- manipulate the view in the Kitting Viewer window by rotating, translating, or zooming
- slow down or speed up the animation without any effect on the robot settings or the metrics
- at any time, save a graphics file with all three windows
- at any time, save a text file with information

from the Metrics & Settings window and the Kitting Command & Messages window

- at any time, start the simulation over from the beginning
- at any time, exit from the Kitting Viewer.

Details of how to use the mouse and keyboard to exercise the functionalities above are described in Section 2.

When the Kitting Viewer is run with simulation:

- It reads a file describing the initial state of the workstation.
- It reads a file describing the goal state of the workstation.
- It reads a robot command file describing robot actions intended to change the workstation from the initial state to the goal state.
- Optionally, it reads a scoring file describing how to combine metrics in order to produce a total score. If the scoring file is not provided, the hard-coded equivalent of a default scoring file is used.
- It follows the user's instruction about whether to treat objects that are the same stock keeping unit as being equivalent when it checks positions.
- Optionally, it uses a position tolerance provided by the user. If no tolerance is provided, a default tolerance is used.
- Until all robot commands are executed, each time the user gives the go-ahead, the execution of another command is simulated graphically, the current state of the workstation is updated, the metrics are updated, and the robot settings are updated.
- After all robot commands are executed, each time the user gives the go-ahead, the position of another movable object in the goal state is checked against its position in the current state. The goal state file does not have to specify the state of all objects, only those objects whose position is important. Also when this stage is reached, additional metrics are calculated and displayed.

When the Kitting Viewer is run without simulation it behaves as described above, except that:

- No robot command file is read.
- A file is read describing the as-built state of the workstation.
- The simulation phase of activity does not occur (since there are no commands to simulate execution of).
- If no scoring file is provided, the hard-coded equivalent of a different default file is used for scoring. This file gives no weight to the command execution metrics since no commands were executed.

2 Running the Kitting Viewer

2.1 Starting The Kitting Viewer

The Kitting Viewer is started by typing a command of one of the two following forms into a terminal window (without any line breaks). Items in angle brackets such as `<commandFile>` should be replaced by literal values. Items in square brackets such as `[-s <scoringFile>]` are optional. The order of pairs of arguments is irrelevant; for example `-i <initFile>` could be first rather than second.

```
kittingViewer -c <commandFile>
 -i <initFile> -g <goalFile>
 -e <equiv> [-s <scoringFile>]
 [-t <tolerance>]
```

```
kittingViewer -b <asBuiltFile>
 -i <initFile> -g <goalFile>
 -e <equiv> [-s <scoringFile>]
 [-t <tolerance>]
```

<commandFile> is the name of a file of canonical robot command language (CRCL) commands. See Section 4 for details of the language.

<asBuiltFile>, <initFile>, and <goalFile> are names of XML kitting workstation files conforming to the XML schema for kitting, kitting.xsd. Details of the kitting.xsd ontology are given in Section 5.

<equiv> must be true or false. True indicates that objects with the same stock keeping unit are equivalent, so it is OK if such objects switch positions from what is given in the goal file. False indicates that each object is unique, so that each object must be where the goal file says it should be.

<scoringFile> is the name of an XML file conforming to scoring.xsd, the XML schema for scoring files. See Section 6 for details.

<tolerance> is location tolerance in millimeters. The default value is 0.2 millimeters.

Example1:

```
kittingViewer -c commands
 -i init.xml -g goal.xml
 -e true -t 1.5
```

Example2:

```
kittingViewer -b built.xml
 -i init.xml -g goal.xml
 -e false -s score.xml
```

2.2 Controlling The Kitting Viewer

Controlling the Kitting Viewer is accomplished by using the mouse and single keys on the keyboard. When the Kitting Viewer starts up, a set of one-line instructions is printed in the terminal window from which the Kitting Viewer was started. Those instructions have the same meaning as the longer explanations given below.

- r (rotate) key -- The r key toggles the behavior of the left mouse button between translating and rotating the picture. This functionality is included because some mice do not have a middle button. Press r once and the left mouse button controls rotation. Press r again and the left mouse button controls translation (the original setting).
- Left mouse button -- By default, the left mouse button is used to translate the picture. Position the cursor anywhere in the Kitting Viewer window, hold down the left mouse button and move the cursor by moving the mouse. The picture will move as though it is glued to the cursor. If the r key has switched the left mouse button to rotation, it behaves like the middle mouse button, as described in the next paragraph.
- Middle mouse button -- The middle mouse button is used to rotate the picture. This is a little trickier than the other two mouse buttons. To rotate the picture, position the cursor inside the window near an edge of the window, hold down the middle mouse button, and move the cursor in a straight line towards the opposite edge. The picture rotates around an axis that is perpendicular to the line of mouse motion. In order to allow for fine positioning, the amount of rotation that occurs for a given amount of mouse motion decreases as the picture is zoomed in. Hence, if you want to turn the picture completely over, it is best to zoom out, rotate, and zoom back in again.
- Right mouse button -- The right mouse button is used to zoom in or out. To zoom out, position the cursor near the bottom of the picture, hold down the right mouse button and push the mouse away from you (moving the cursor up); that appears to push the picture away from you. To zoom in, position the cursor near the top of the picture, hold down the right mouse button and pull the mouse toward you (moving the cursor down); that appears to pull the picture toward you. Moving the mouse side

to side while holding down the right mouse button does nothing. There are limits to how far in or out you can zoom. At the highest magnification, it is easy to see a separation of half a millimeter. This is zooming, not moving the point of view, so the eye never goes through the picture.

- `f` (faster) key -- If the `f` key is pressed, the animation speeds up. Pressing `f` twice doubles the speed. The programmed speed of the robot is not changed. The speed-up is done by changing the speed of a pseudo elapsed time clock. Changing the speed of the animation has no effect on the metrics and settings (including the execution time).
- `s` (slower) key -- If the `s` key is pressed, the animation slows down. Pressing `s` twice cuts the speed in half. See the description of the `f` key for details.
- `v` (view) key -- If the `v` key is pressed, the view in the Kitting Viewer window returns to its original position.
- `e` (execute) key -- If the `e` key is pressed when the plan is not completely executed and no action command is executing, the next command in the plan is executed and all windows are updated. If the `e` key is pressed when the plan is completely executed, but not all movable goal objects have had their positions checked, the position of the next movable goal object is checked, the metrics are updated and a message is printed in the Command & Messages window. If the `e` key is pressed when the positions of all movable goal objects have been checked or when an action command is in progress, nothing happens.
- `d` (dump) key -- If the `d` key is pressed, a combined image of all three windows is saved in a file. The name of the file will be `kittingViewer_`*`N`*`.ppm`, where *`N`* starts at 0000 and increases by 1 each time the `d` key is pressed. The ppm (portable pixmap) format is a common graphics format that many graphics utilities can handle.
- `t` (text) key -- If the `t` key is pressed, the text in the Metrics & Settings window followed by the text in the Command & Messages window is saved in a text file. The name of the file will be `kittingViewer_`*`N`*`.txt`, where *`N`* starts at 0000 and increases by 1 each time the `t` key is pressed.
- `D` (Dump) key -- If the `D` key is pressed, both an image file and a text file are saved, as if both the `d` key and the `t` key had been pressed. Both files will have the same value of *`N`* in the file name.
- `a` (again) key -- If the `a` key is pressed, the animation starts over again from the beginning. Everything except the viewing parameters is reset.
- `Esc` (escape) key -- If the `Esc` key is pressed, the Kitting Viewer program exits, and the windows disappear.

2.3 Errors

In order to fully evaluate an input file, many file errors are noted, but the Kitting Viewer continues to run. Some errors are too severe for the Kitting Viewer to handle. For these, it prints an error message and exits.

When the parser encounters a line that it cannot parse, it adds an `UnreadableMsg` to the list of commands it has parsed. The `UnreadableMsg` includes the text of the line on which the parse error occurred. When the `UnreadableMsg` is executed, the value of parse errors is increased by one and the `UnreadableMsg` is displayed in the Kitting Command & Messages window so the user can see the line that caused the problem.

3 Metrics

All of the metrics used in the Kitting Viewer are numbers. All but one of the metrics are objective and require no human judgement. The final metric, score, is a subjective combination of the other metrics in which the other metrics are weighted and combined as specified in a scoring file selected (and possibly built) by the user. Scoring details are given in Section 6.

Most metrics are calculated and displayed during both phases of Kitting Viewer operation, but some a shown only during the second phase.

3.1 First Phase Metrics

The following metrics implemented in the Kitting Viewer are evaluated in the first phase of Kitting Viewer operation at the end of each CRCL command, with cumulative values.

- action commands executed successfully — the number of action commands that have been executed so far. An action command is any command that takes time to execute.
- other commands executed successfully — the number of commands that are not action commands that have been executed so far. These are mostly setting commands. Executing these commands is assumed to take a negligible amount of time.
- total robot distance moved — the total distance that the tool tip has moved so far. This is calculated as the total of the distances between points in the move commands, taken in order (and starting at the place where the controlled point is located initially). If a move with multiple legs is commanded, this metric is updated as the end of each leg is reached.
- total execution time — the total time taken so far by executing action commands. This does not include any time that may elapse between when one command finishes execution and when the user tells the system to execute another command. The total execution time is meant to be very close to the actual amount of time that would be taken by an actual kit-building system without user intervention.
- useless commands executed — the number of commands that have been executed so far that do not change the state of the workstation. Such commands have no effect, so they are useless. They are not counted as errors, but the scoring system may penalize executing them. Examples of such commands include a `CloseGripper` command given when the gripper is already closed and an `OpenToolChanger` command given when the tool changer is already open.
- range errors — the number of times a command tries to set a parameter to a value that is out of the allowed range of the parameter.
- parse errors — the number of lines in the CRCL command file that cause an error in the command file parser.
- command sequence errors — the number of commands that are out of sequence. An `InitCanon` command is out of sequence if it is not the first command in the file. An `EndCanon` command is out of sequence if it is not the last command in the file. Other commands are out of sequence if they occur before `InitCanon` or after `EndCanon`.
- gripper use errors — the number of `OpenGripper` and `CloseGripper` commands that cannot be executed because the robot is not holding a gripper.
- tool change errors — the number of `OpenToolChanger` and `CloseToolChanger` commands that cannot be executed. A tool change error occurs, for example, if an attempt is made

to put a gripper into a tool holder that already holds another gripper.

- motion errors — the number of commands that attempt to move the robot improperly. A motion error occurs, for example, if a motion command is given that reorients the robot when it is not capable of being reoriented (whenever no gripper is mounted on the robot).
- total errors — the sum of the range errors, parse errors, command sequence errors, gripper use errors, tool change errors, and motion errors.

3.2 Second Phase Metrics

After all commands have been executed so that all objects are in their final positions, in the second phase of operation, the Kitting Viewer goes through the goal objects one at a time and evaluates the following metrics cumulatively. Metrics from the first phase continue to be displayed, but except for Total Errors, are not changed. The user's choice of whether to treat objects that have the same SKU as interchangeable is used in computing the first two of these metrics.

- Objects Located Correctly — the number of movable goal objects checked so far that were moved from their initial location to the correct goal location.
- Object Location Errors — the number of movable goal objects checked so far whose final position is not the one given in the goal file.
- Total Basic Goal Object Distance Moved — the total net distance moved from initial location to final location by movable basic goal objects checked so far. A movable basic goal object is a movable solid object in the goal file that cannot be decomposed into other solid objects. For example, a Part is a basic object, but a Kit is not (since it may be decomposed into a KitTray and Parts).
- Total Errors — the number of Total Errors from the first phase plus the number of Object Location Errors for movable goal objects checked so far.
- Score — the score so far, as described in Section 6.

Because the CRCL commands include only primitive commands such as `OpenGripper`, `CloseGripper`, and `Move` and do not include task commands such as `PickUpObject` and `MoveObject`, the metrics do not include any measures of success or failure at picking up objects or moving them. For example, if the robot moves to a suitable position for picking up an object and closes the gripper, but the gripper is not suitable for picking up the object, the object is not picked up, but no error is recorded. In such cases, an error will almost certainly be found later in the position of the object that was not moved; in addition, the scoring system is likely to penalize the wasted motion.

3.3 Metrics Not Included

Some other metrics have been considered but are not included in the Kitting Viewer, as follows.

- number of collisions — This is not included because it is very difficult to calculate. A solid modeler, plus a sophisticated set of calls to the modeler would be required to find collisions automatically and reliably. Collisions can be identified by the user by watching the simulation and seeing if solid objects ever collide.
- number of robot motions outside the robot's work volume — This could be implemented without too much difficulty. The robot's work volume is specified in the init file.
- number of objects the gripper tries to attach to too heavy for the gripper to lift — This is not calculated because the CRCL commands include only primitive

commands such as `OpenGripper`, `CloseGripper` and `Move` and do not include task commands such as `PickUpObject` and `MoveObject`. Hence, whether the robot is trying to attach to something is open to subjective judgement.

- number of objects too heavy for the robot to lift but attached to the gripper when the robot moves — This could be implemented without too much difficulty. The weights of the gripper and all objects that have a stock keeping unit are specified in the init file.

4 Canonical Robot Commands Language

It is desirable that numerous commercial robot systems be able to immediately execute the plan for the series of actions required to transition from the initial state to the goal state of the kitting problem. However, there is currently no accepted standard robot programming language. For this reason, the authors have developed a canonical robot control language (CRCL) that attempts to be a lowest common denominator of robot programing languages. It is anticipated that kitting plans can be translated into CRCL command sets which may then be evaluated by standardized metric software. The CRCL command sets may then be translated into a specific robot platform's language.

The syntax of commands is given below using C++ syntax. The command name is given followed by the command arguments (if any) in parentheses, including the types of the arguments.

Note that the robot cannot be commanded by canonical robot commands in terms of its joint angles (or distances).

Three of the CRCL commands use the Pose structure. The Pose structure gives the location and orientation of the coordinate system of the controlled object in the units of the current operating coordinate system. The controlled object is the gripper if the robot has one attached or the outermost component of the robot arm if not. The location is specified by the point in current operating coordinates at which the origin of the coordinate system of the controlled object lies. The point is described by giving its X, Y, and Z values. The orientation of the controlled object is specified by giving the I, J, and K components in current operating coordinates of the Z and X axes of the coordinate system of the controlled object.

The complete list of CRCL commands follows.

`CloseGripper()`
Close the gripper.

`CloseToolChanger()`
Close the tool changer on the robot so that it attaches to a tool. The robot must be in an appropriate position with respect to the tool for the changer mechanism on the robot to attach to the tool.

`Dwell (double time)`
Stay motionless for the given amount of `time` in seconds.

`EndCanon(int reason)`
Do whatever is necessary to stop executing canonical robot commands. No specific action is required. The robot controller should not execute any canonical robot command except `InitCanon` after executing `EndCanon` and should signal an error if it is given one. This command will normally be given when execution of a plan is complete. It may also be given if the plan interpreter detects an error in the plan or is unable to proceed for any other reason. A value of 0 for `reason` indicates that execution of a plan has completed successfully. A positive value of `reason` indicates not.

`InitCanon()`
Do whatever is necessary to get ready to move.

Length units, angle units, and operating coordinate system are set to the default units. This command will normally be given when the plan interpreter opens a plan to be executed.

`Message (string message)`
Display the given `message` on the operator console.

`MoveStraightTo(Pose * pose)`
Move the controlled point in a straight line from the current pose to the given `pose`, and stop there.

`MoveThroughTo (Pose ** poses, int numPoses)`
Move the controlled point along a trajectory passing near all but the last of the given `poses`, and stop at the last of the given `poses`. The `numPoses` gives the number of `poses`.

`MoveTo(Pose * pose)`
Move the controlled point along any convenient trajectory from the current pose to the given `pose`, and stop there.

`OpenGripper()`
Open the gripper.

`OpenToolChanger()`
Open the tool changer on the robot so that it releases the end effector. This is normally done after the end effector attached to the robot has been moved into an end effector changer.

`SetAbsoluteAcceleration (double acceleration)`
set the acceleration for the controlled point to the given value in length units per second per second.

`SetAbsoluteSpeed(double speed)`
Set the speed for the controlled point to the given value in length units per second.

`SetAngleUnits(string UnitName)`
set angle units to the unit named by the `UnitName`. The `UnitName` must be one of degree or radian. All commands that use angle units (for orientation or orientation tolerance) are in terms of those angle units. Existing values for orientation are converted automatically to the equivalent value in new angle units. The default angle unit is degree.

`SetCoordinateFrame (string CoordSystem)`
set the operating coordinate system to the system referred to by `CoordSystem`. The `CoordSystem` must be one of `Workstation`, `RobotBase`, or `ToolTip`.

`SetEndAngleTolerance (double tolerance)`
Set the tolerance for the orientation of the end of the arm (whenever there is no gripper there) or of the gripper (whenever a gripper is on the end of the arm) to the given value in current angle units. This applies to the X-axis direction and the Z-axis direction.

`SetEndPointTolerance (double tolerance)`
Set the tolerance for the position of the end of the arm (whenever there is no gripper there) or of the tool centre point (whenever a gripper is on the end of the arm) to the given value in current length units.

`SetIntermediatePointTolerance (double tolerance)`
Set the tolerance for smooth motion near intermediate points to the given value in current length units.

`SetLengthUnits(string UnitName)`
Set length units to the unit named by the `UnitName`. The `UnitName` must be one of `inch`, `mm` or `meter`. All commands that use length units (for location, tolerance, speed, and acceleration) are in terms of those length units. Existing values for speed, position, acceleration, etc. are converted automatically to the equivalent value in new length units. The default length unit is millimeters, `mm`.

`SetRelativeAcceleration (double percent)`

Set the acceleration for the controlled point to the given percentage of the robot's maximum acceleration.

`SetRelativeSpeed (double percent)`
`Set the speed for the controlled point to the given percentage of the robot's maximum speed.`

`StopMotion(integer isEmergency)`
Stop the robot motion. If `isEmergency` is not 0, then stop as soon as possible regardless of damage to the system. If `isEmergency` is 0 then come to a graceful stop.

A file format for representing CRCL commands has been devised. Appendix A shows an example of a file prepared using this format. A C++ class model of CRCL commands has been built, and a parser has been built in C++ for reading CRCL files and populating CRCL class instances.

5 State Representation

5.1 Kitting Workstation Model

The project in which the Kitting Viewer was developed has modeled the state of a kitting workstation as semantically identical ontologies in Web Ontology Language (OWL) [11], [12], [13],and XML schema definition language (XSDL) [9], [10], [8].

Figure 2 shows a diagram of the top level of the model. The complete XML schema model is given without the in-line documentation in Appendix B (arranged alphabetically for easy reference). With the documentation nodes, the model is about twice as long as in Appendix B. The OWL model is about half again as long as that.

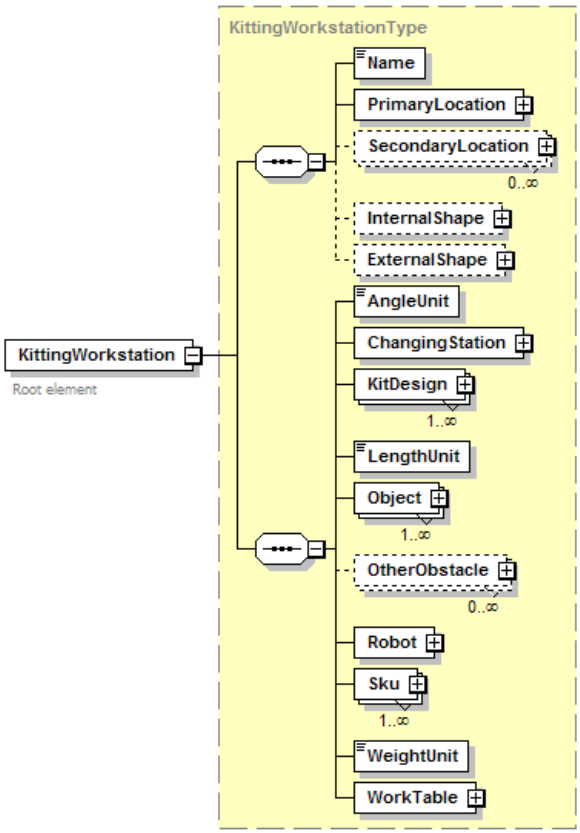

Figure 2. Workstation Model

This paper explains only selected features of the model. Most of the principal items in a workstation appear in Figure 2. One important item that does not appear explicitly is grippers, which are either on the robot or in the changing station. Other important item types not shown include large box with empty kit trays, large box with kits, parts tray with parts, large container, parts tray, kit tray, kit, and part. Those would all be found as instances of the Object element in the figure, or as items inside them.

Two types, DataThingType and SolidObjectType, are at the top level of the type hierarchy used in the ontology. All other types derive from one of these two. Both DataThingType and SolidObjectType have a Name element. Hence, everything that is not a simple type such as a string or a number has a Name. Solid objects have a primary position

and zero to many secondary positions. All the positions of a solid object represent the same location in space. Each position of a solid object B specifies another solid object with respect to which B is located. The workstation is the only solid object allowed to be located with respect to itself. Data things do not have any position. Each solid object has a native coordinate system. The location of the coordinate system of the object is what is specified by the position information.

In the Kitting Viewer, the first secondary position of each solid object is its position relative to the workstation. This is the position used to draw the object. The primary position of each object is specified hierarchically. For example, when the robot is using a gripper and moving a finished kit, a part in the kit is positioned relative to the kit, the kit is positioned relative to the gripper, the gripper is positioned relative to the robot, and the robot is positioned relative to the workstation.

Each solid object may have an InternalShape element of InternalShapeType and/or an ExternalShape of ExternalShapeType. Internal shapes are those that are part of the model. External shapes are those described in a shape file built in some other system. Currently, the only derived type of internal shape is BoxyShapeType. The Kitting Viewer is limited to drawing boxy shapes (or assemblies of boxy shapes) plus the robot and grippers, whose shapes are hard-coded.

The design of a kit is modeled as shown in Figure 3. In addition to a Name (inherited from DataThingType), a KitDesignType has a KitTraySkuName which gives the name of the stock keeping unit (SKU) of the kit tray to be used in the kit, and one to many PartRefAndPose, each of which gives the name of the SKU for a part and the location of the part relative to the kit tray.

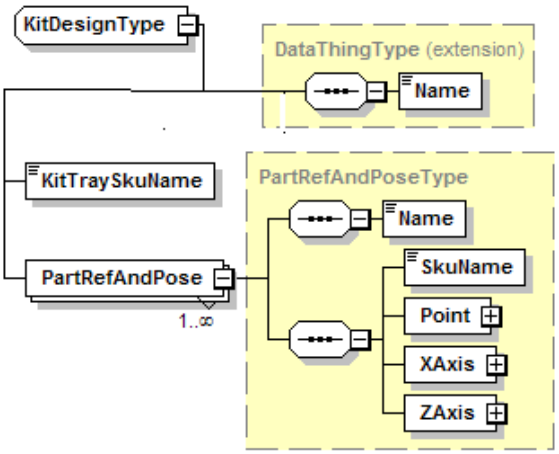

Figure 3. Kit Design Model

5.2 Updating the Model

When the Kitting Viewer is used to simulate executing a CRCL file, it does a great deal of modeling in the first phase of running. When the initial state file is read, a model of the initial state is built and saved as the current state of the workstation. A model of the goal state is also built and saved. As the Kitting Viewer runs and CRCL commands are executed, the Kitting Viewer determines the effect of executing each command on the current state and updates it. The second phase of Kitting Viewer operation compares the evolved current state with the goal state. When the Kitting Viewer is given an as-built file, it goes directly to the second phase.

The logic of state changes is complex in some cases. The most interesting cases involve what to do when executing `CloseGripper` and `OpenGripper` commands. Details for `CloseGripper` are given below.

A key issue is that composite objects may go out of existence or come into existence while kits are being built. A kitting workstation builds kits. The kits do not exist in the initial state but they do exist in the goal state, so it is necessary to make them start to exist at some point in the process. The initial state includes parts trays with parts. When all the parts are

removed from a parts tray with parts, it goes out of existence. Of course the parts tray remains, but it is no longer a component of a parts tray with parts. In the Kitting Viewer, a kit comes into existence when the first part is placed in a kit tray, and a parts tray with parts goes out of existence when the last part is removed from it.

For `CloseGripper`, a great deal of processing is needed to determine whether the gripper attaches to something when the command is executed. This is done as follows.

If all of the following hold:

1. The robot is holding an end effector.

2. The end effector is a single cup vacuum gripper (that's the only kind of gripper the Kitting Viewer knows how to use).

3. Either:

3a. There is a parts tray or kit tray with a topless boxy shape such that the gripper cup is within the Kitting Viewer's set tolerance of the bottom of the tray and is within 1 mm of the XY location of the origin of the tray. OR

3b. There is a part with a boxy shape with top such that the gripper cup is within the set tolerance of the top of the part and is within 1 mm of the XY location of the middle of the top of the part.

4. The gripper is able to pick up that type of part or tray.

5. The Z axis of the gripper is 0,0,-1 and the Z axis of the object is 0,0,1 (both within the tolerances for vectors).

6. The gripper is open (implying the gripper is not holding anything).

Then the gripper will attach to an object. Call it B.

- If 3b above occurred, then B is a part.
- If 3a above occurred with a parts tray in a parts tray with parts, then B is the parts tray with parts.
- If 3a above occurred with a parts tray not in a parts tray with parts, then B is the parts tray.
- If 3a above occurred with a kit tray in a kit, then B is the kit.
- If 3a above occurred with a kit tray not in a kit, then B is the kit tray.

When the gripper attaches to an object, the primary state changes (i.e. changes in states present in the model) are that the gripper is closed, and the pose of the object is changed so that the object is located relative to the gripper. In addition, as described above, if the object is the last part in a parts tray with parts, the parts tray with parts will go out of existence. The Kitting Viewer has several other state variables for positions to make its work more efficient, and these are also updated.

For `OpenGripper` the processing is also complex. When an object B is released, its position is no longer relative to the gripper. The Kitting Viewer has to find another object for B's position to reference and set the numerical values for the position. In addition, if a part is put into an empty kit tray, a kit must be brought into existence.

5.3 XML <-> OWL Data File Translation

In order to make it possible to use OWL kitting instance data files for initial state, goal state, and as-built state files to drive the Kitting Viewer, it is planned to build a translator that will translate an OWL kitting instance file conforming to kittingClasses.owl into an XML data file conforming to kitting.xsd. As mentioned earlier, kittingClasses.owl and kitting.xsd represent the same ontology in different languages, so this translation is possible in principle.

A translator named xml2owl has already been built that does the reverse job. It translates an XML data file conforming to kitting.xsd into an OWL instance file conforming to

kittingClasses.owl. The easiest way to build a conforming OWL kitting instance file is to build the corresponding XML data file and run it through xml2owl. This is easier than using an OWL editing system for three reasons. First, the XML file is less than half the size of the OWL file. Second, the XML file has many fewer statements than the OWL file. Third, and most importantly, the kitting ontology assumes a closed world. Any names of types, attributes, etc. not found in kitting.xsd that appear in a data file are errors. Readily available XML editing tools will find such errors automatically. These editors also assume a closed world. OWL and OWL editors use an open world approach in which errors such as typing names incorrectly in an instance file are not regarded as errors. Hence, it is impossible in principle to debug an OWL instance file automatically. For large instance files, manual debugging is also impossible in practice.

It also appears to be feasible to built a generic XSDL to OWL translator for straightforward XML schemas such as kitting.xsd. That translator would, for example, be able to automatically produce the OWL kitting ontology file, kittingClasses.owl, from the XML schema file, kitting.xsd.

6 Scoring

Calculating a single numeric score for a kit building plan that is a CRCL file is accomplished by having a user name a scoring file when the Kitting Viewer is started. The scoring file specifies (1) a weight for each of several factors (2) whether each weight is multiplicative or accumulative, and (3) a valuation function for producing a number between 0 and 1 from each factor. The valuation function is optional if the factor value is sure to be between 0 and 1. The Kitting Viewer combines partial scores as described below to form a single score that may be used for comparison.

The score is produced only during the second phase of operation, after all CRCL commands have been executed and the positions of goal objects are being checked. During this phase, the factors and the score are recomputed after each movable goal object is checked.

6.1 Factors

To compute a score, the Kitting Viewer first combines the metrics into the following five factors:

- Right Stuff — The Right Stuff value, *R*, is a measure of achieving goal positions. Let *G* be the number of objects in the goal file placed correctly so far, *B* be the number of objects in the goal file placed incorrectly so far, and *N* be the number of objects in the goal file checked so far. Then
  $R = (G - B)/N$.
  If *R* is less than zero, it is set to zero.
- Command Execution — The Command Execution value, *C*, is the fraction of all commands in the command file that were executed correctly. This factor does not change during the second phase of Kitting Viewer operation. Let *K* be the total number of commands executed successfully and *E* be the number of errors that are not location errors.
  Then $C = K/(K + E)$.
- Distance — The Distance value, *D* is a measure of efficiency of robot motion in terms of distance. Let *J* be the total distance moved from initial position to goal position by all basic goal objects that have been checked so far, *L* be the total distance moved by the robot, and *F* be the fraction of movable objects that have been checked so far. Then $D = (2 \times J)/(L \times F)$. If *D* is greater than 1, it is set to 1. The numerator, $(2 \times J)$, is a crude measure of a short distance to move the robot in order to move the objects that have been moved so far to their goal positions.

- Time — The Time value, *T* is a measure of efficiency of robot motion in terms of time. The teleport time, *P*, is a crude measure of a fast time for moving the objects that have been moved so far to their goal positions. Let *J* and *F* be as in the preceding item, and let *Q* be the maximum speed of the robot. Then $P = (2 \times J)/Q$. Let *H* be the total execution time. Then $T = P/(H \times F)$. If *T* is greater than 1, it is set to 1.
- Useless Commands — The Useless Commands factor is the value of the useless commands metric. This factor does not change during the second phase of Kitting Viewer operation.

6.2 Scoring File

A scoring file used in the Kitting Viewer is an XML data file conforming to the scoreKitting.xsd XML schema file. The schema file is shown in Appendix C with the documentation (which is about three-quarters of the file) removed.

The scoring file shown in Figure 4, which conforms to the scoreKitting.xsd schema, is the one whose hard-coded equivalent is used in the Kitting Viewer if the user does not designate a scoring file.

The scoring file, as prepared by a user, designates each of the five factors as being either multiplicative or additive. For additive factors, a weight may be assigned in the file. For each factor, a valuation function may be assigned. As described in more detail below, a valuation function takes a raw factor with an arbitrary range and produces a number between 0 and 1. The final stage of producing a score combines numbers between 0 and 1. In the scoring file shown in Figure 4, all five factors have equal weights, rightStuff is multiplicative while the other four factors are additive, and there is a valuation function only for uselessCommands.

```
<?xml version="1.0"?>
<scoreKitting
 xmlns="urn:Kitting"
 xmlns:xsi=
  "http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation=
  "urn:Kitting ../schema/scoreKitting.xsd">
 <rightStuff>
  <isAdditive>false</isAdditive>
  <weight>1</weight>
 </rightStuff>
 <commandExecution>
  <isAdditive>true</isAdditive>
  <weight>1</weight>
 </commandExecution>
 <distance>
  <isAdditive>true</isAdditive>
  <weight>1</weight>
 </distance>
 <time>
  <isAdditive>true</isAdditive>
  <weight>1</weight>
 </time>
 <uselessCommands>
  <isAdditive>true</isAdditive>
  <weight>1</weight>
  <valueFunction>
   <bestValue>0.000000</bestValue>
   <width>0.000000</width>
   <taper>3.000000</taper>
   <taperSide>plus</taperSide>
  </valueFunction>
 </uselessCommands>
</scoreKitting>
```

Figure 4. Scoring File

6.3 Valuation Functions

A valuation function takes any number as its argument and returns a number between 0 and 1.

If a raw factor (useless commands, for example) is not necessarily between 0 and 1, a valuation function must be assigned to that factor. If a raw factor is always between 0 and 1 (right stuff, for example), a valuation function may be assigned, but is not required.

The idea behind valuations functions is that for

most raw factors there is a single value or a range of values that is desirable, and there may be values near the desirable range that are still good, but are less good.

Valuation functions are specified using four parameters; `bestValue`, `width`, `taper`, and `taperSide`. The first three are numbers; `taperSide` is one of three strings: `minus`, `plus`, or `both`. The `width` parameter is used only if the `taperSide` is `both`, and is the width of a range with the `bestValue` in the middle for which the valuation function should return 1. The taper value must be zero or positive and is the range distance over which the returned value should taper from 1 to 0. Figure 5 shows three examples of valuation functions using a variety of parameters. The raw factor value is on the horizontal axis and the returned value on the vertical axis.

6.4 Score Computation

The scoring system is designed so that the score it produces is always between 0 and 100.

Each factor is designated as additive or multiplicative. A factor value Vi between 0 and 1 is found for each additive factor and a factor value Ui between 0 and 1 is found for each multiplicative factor. Each additive factor is assigned a non-negative weight Wi. An additive score Sa is produced by multiplying each additive value by its weight, adding the products together, and dividing by the sum of the weights. If there are no additive factors or their weights are all zero, Sa = 1. Weights assigned to multiplicative factors are not used.

$$S_a = \frac{(V_1 \times W_1) + (V_2 \times W_2) + \ldots + (V_n \times W_n)}{W_1 + W_2 + \ldots + W_n}$$

The value of S_a will be between 0 and 1 since all the components of the equation are positive and the largest the numerator can be is the size of the denominator. Then the total score S is found by finding the product of S_a, 100, and all the multiplicative factors.

$$S = (100 \times S_a \times U_1 \times U_2 \times \ldots \times U_m)$$

.

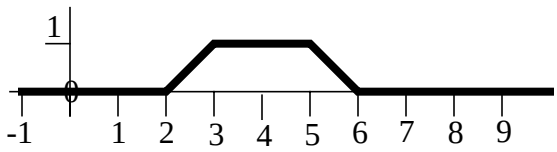

Figure 5A.
`bestValue = 4  width = 2`
`taper = 1  taperSide = both`

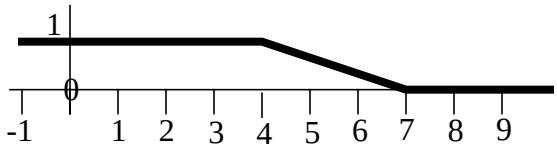

Figure 5B.
`bestValue = 4`
`taper = 3  taperSide = plus`

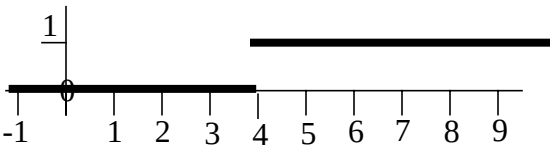

Figure 5C.
`bestValue = 4`
`taper = 0  taperSide = minus`

Figure 5. Valuation Functions

7 Software

7.1 Kitting Viewer Software

The Kitting Viewer software that is compiled together to make the Kitting Viewer is in C++ [7] using the OpenGL Library [6] for graphics and the Boost “regex” regular expression library [2]. All the software is system independent, so that it should be straightforward to compile it on any platform.

The OpenGL and Boost libraries are freely available for most platforms. The authors have compiled the Kitting Viewer only on Linux so far.

The source code for the Kitting Viewer includes 13 .cc files totalling about 25,000 lines, plus header (.hh) files. These are compiled together to make the Kitting Viewer executable. Most of the hand-written files have extensive in-line documentation.

Several of the files were generated automatically by a generator built by one of the authors that reads an XML schema and writes C++, YACC, and Lex files (called simply the "generator" below). In some cases the automatically generated files were modified by hand to be more useful in the Kitting Viewer. The automatically generated files have little in-line documentation.

7.1.1 commandParser.cc

#includes
commandParser.hh
This hand-written file is a CRCL command file parser.

7.1.2 canonicalMsgView.cc

#includes
canonicalMsg.hh
This hand-written file implements the "process" function of each CRCL command by having it put its text representation into the Kitting Viewer's command string array.

7.1.3 mouse.cc

#includes:
mouse.hh
This hand-written file implements the graphics manipulation functionality of the mouse.

7.1.4 kittingViewer.cc

#includes:
canonicalMsg.hh
kittingClassesView.hh
commandParser.hh
kittingViewer.hh
view.hh
xmlSchemaInstance.hh
This 6600-line hand-written file includes the main function of the Kitting Viewer, the instructions to open, close and read files, the picture construction, the location checking, the score computation, the transformation math, the user messages, the command execution, the pseudo-elapsed time clock, the generation of colors, the animation calculations, and the state maintenance.

7.1.5 viewKitting.cc

#includes:
mouse.hh
view.hh
This hand-written file calls OpenGL graphics functions, creates, displays, and reshapes windows, handles the mouse and keyboard, and dumps graphics and text files.

7.1.6 kittingClassesView.cc

#includes
kittingClassesView.hh
kittingYACC.hh
xmlSchemaInstance.hh
This 3600-line file started as a 3200-line automatically generated file (produced by the generator) named kittingClasses.cc. The remaining lines were added by hand. The automatically generated portion implements printSelf functions (that are not called in the Kitting Viewer) and the constructors for all the C++ classes derived from kitting.xsd. These constructors are called by the actions in kittingYACC.cc

7.1.7 kittingLex.cc

#includes
source/kittingYACC.hh
xmlSchemaInstance.hh
This file implements the lexical scanner used by the parser that reads in the initial, goal, and as-built files. It was generated automatically from the Lex [5] file kitting.lex by the flex [4] utility. The kitting.lex file was generated

automatically from kitting.xsd by the generator.

7.1.8 kittingYACC.cc

#includes
kittingYACC.hh
kittingClassesView.hh
xmlSchemaInstance.hh
This file implements the parser that reads in the initial, goal, and as-built files. It was generated automatically from the YACC [5] file kitting.y by the bison [3] utility. The kitting.y file was generated automatically from kitting.xsd by the generator.

7.1.9 Pose.cc

#includes
Pose.h
This small hand-written file normalizes vectors and checks orthogonality of axes.

7.1.10 scoreKittingClasses.cc

#includes
scoreKittingClasses.hh
xmlSchemaInstance.hh
This file implements printSelf functions (that are not called in the Kitting Viewer) and the constructors for all the C++ classes derived from scoreKitting.xsd. These constructors are called in the scoreKittingYACC.cc file. The scoreKittingClasses.cc file was generated automatically from scoreKitting.xsd by the generator.

7.1.11 scoreKittingLex.cc

#includes
scoreKittingYACC.hh
This file implements the lexical scanner used by the parser that reads in the scoring file. It was generated automatically from the scoreKitting.lex file by the flex utility. The scoreKitting.lex file was generated automatically from scoreKitting.xsd by the generator.

7.1.12 scoreKittingYACC.cc

#includes
scoreKittingYACC.hh
scoreKittingClasses.hh
xmlSchemaInstance.hh
This file implements the parser that reads in the scoring file. It was generated automatically from the scoreKitting.y file by the bison utility. The scoreKitting.y file was generated automatically from kitting.xsd by the generator.

7.1.13 xmlSchemaInstance.cc

#includes
xmlSchemaInstance.hh
This hand-written file implements classes representing XML built-in types (e.g., IDREF and unsignedInt). Instances of these types may appear in XML data files conforming to an XML schema.

7.2 Parser Software

Three stand-alone parsers have been built in connection with the Kitting Viewer in order to check data files used by the Kitting Viewer.

A kittingParser has been built to check the initial state, goal state, and as-built files, all of which conform to the kitting.xsd XML schema. It is built from a 1-page kittingParser.cc file containing a main function plus the kittingYACC.cc, kittingLex.cc, kittingClasses.cc, and xmlSchemaInstance.cc source code files.

A scoreKittingParser has been built to check scoring files which conform to the scoreKitting.xsd XML schema. It is built from a 1-page scoreKittingParser.cc file containing a main function plus the scoreKittingYACC.cc, scoreKittingLex.cc, scoreKittingClasses.cc, and xmlSchemaInstance.cc source code files.

The kittingParser and scoreKittingParser are both compiled with a switch turned on that causes the lexical scanner to echo everything it reads. In these stand-alone parsers, if there is a parsing error, parsing will stop at the point in

the file where the error occurred, and the file will have been printed on the monitor up to that point so that the error is easy to locate. The compiler switches are off when the Kitting Viewer is compiled. A parse error from these parsers in the Kitting Viewer will cause it to stop, but the file being read will not have been echoed, so there will be no clue regarding the location of the error.

A commandParser has been built to check CRCL command files. The source code consists of the canonicalMsg.cc file, a half-page commandParserMain.cc file, and the commandParser.cc file. The canonicalMsg.cc file implements the "process" function of each CRCL command by having it print its text representation on the monitor.

8 Conclusion

The Kitting Viewer has a number of useful capabilities, but could certainly be improved.

As mentioned earlier, it would be useful to be able to drive the Kitting Viewer using OWL instance files. Efforts to implement that by building an OWL to XML translator are already underway.

The Kitting Viewer's rules for the conditions under which a gripper can pick up an object are quite limited. It would be useful to expand these rules. In particular, it would be useful to make use of grasp pose information for solid objects. Grasp poses are already part of the model but are not being used.

The range of shapes that the Kitting Viewer is able to draw needs to be expanded. More types of internal shape should be added to the model (starting with shapes for grippers) and used. Implementing the ability to use at least one file format for external shapes would also be useful.

The model of a robot includes its acceleration, but that information is not being used in determining the time it takes for the robot to move from one place to another. Only speed is being used. The simulation would be improved by using acceleration as well as speed.

It would be useful to create a large box with kits when a kit is first placed in a large container, and to destroy a large box with empty kit trays when the last kit tray is removed. Currently those two large box types exist whether they live up to their names or not.

It would be good to check robot moves against the robot work volume. The work volume is defined in the model as the union of a set of box shapes. Implementing a check for the way points and end points of each move would not be too difficult.

Scoring files are difficult to edit because they are somewhat difficult to understand. It would be good to have a graphical tool for editing scoring files.

A graphical user interface for starting the kitting viewer would also be useful since there are so many parameters. Some work on this was done for an earlier version of the Kitting Viewer.

Bibliography

1. Balakirsky, S., Kramer, T., and Pietromartire, A., *Metrics and Test Methods for Industrial Kit Building*, National Institute of Standards and Technology, Gaithersburg, MD, USA, NISTIR to appear, 2012
2. http://www.boost.org/doc/libs/1_47_0/libs/libraries.htm
3. http://www.gnu.org/software/bison/
4. http://www.gnu.org/software/flex/
5. Levine, J., Mason, T., and Brown, D., *lex & yacc*. O'Reilly, Cambridge, MA, USA.
6. Shreiner, D., Neider, J., Woo, M., and Davis, T., *OpenGL Programming Guide*. Fourth edition. Addison Wesley, New York, NY, USA.

7. Stroustrup, B., 2000. *The C++ Programming Language*. Addison Wesley, New York, NY, USA.
8. Walmsley, P., 2002. *Definitive XML Schema*. Prentice Hall, Upper Saddle River, NJ, USA.
9. W3C, 2004. *XML Schema Part 0: Primer Second Edition*. In http://www.w3.org/TR/xmlschema-0/.
10. W3C, 2004. *XML Schema Part 1: Structures Second Edition*. In http://www.w3.org/TR/xmlschema-1/.
11. W3C, 2012. *OWL 2 Web Ontology Language Document Overview.* In http://www.w3.org/TR/owl-overview/.
12. W3C, 2009. *OWL 2 Web Ontology Language Primer.* In http://www.w3.org/TR/owl2-primer/.
13. W3C, 2009. *OWL 2 Web Ontology Language Structural Specification and Functional Syntax*. In http://www.w3.org/TR/owl2-syntax/.

Appendix A [Robot Program

```
InitCanon()
SetLengthUnits("meter")
SetAbsoluteSpeed(1.0)
SetRelativeAcceleration(80.0)
SetEndPointTolerance(0.002)
SetIntermediatePointTolerance(0.01)
OpenGripper()
MoveThroughTo({ {{1.5,1,1}, {0,0,-1}, {1,0,0}},
  {{1.5,1,0.0001}, {0,0,-1}, {1,0,0}} }, 2)
CloseGripper()
MoveThroughTo({ {{1.5,1,1}, {0,0,-1}, {1,0,0}},
  {{4,1,1}, {0,0,-1}, {1,0,0}},
  {{4,1,0.5001}, {0,0,-1}, {1,0,0}} }, 3)
OpenGripper()
MoveThroughTo({ {{4,1,1}, {0,0,-1}, {1,0,0}},
  {{8.25,1,1}, {0,0,-1}, {1,0,0}},
  {{8.25,1,0.4}, {0,0,-1}, {1,0,0}} }, 3)
OpenToolChanger()
MoveThroughTo({ {{8.25,1,1}, {0,0,-1}, {1,0,0}},
  {{8.75,1,1}, {0,0,-1}, {1,0,0}},
  {{8.75,1,0.5}, {0,0,-1}, {1,0,0}} }, 3)
CloseToolChanger()
MoveThroughTo({ {{8.75,1,1}, {0,0,-1}, {1,0,0}},
  {{5.659,1.1,1.8}, {0,0,-1}, {1,0,0}},
  {{5.659,1.1,0.1501}, {0,0,-1}, {1,0,0}} }, 3)
CloseGripper()
MoveThroughTo({ {{5.659,1.1,0.5}, {0,0,-1}, {1,0,0}},
  {{3.86,1.07,1}, {0,0,-1}, {1,0,0}},
  {{3.86,1.07,0.6501}, {0,0,-1}, {1,0,0}} }, 3)
OpenGripper()
MoveThroughTo({ {{3.86,1.07,1}, {0,0,-1}, {1,0,0}},
  {{5.659,0.9,0.5}, {0,0,-1}, {1,0,0}},
  {{5.659,0.9,0.1501}, {0,0,-1}, {1,0,0}} }, 3)
CloseGripper()
MoveThroughTo({ {{5.659,0.9,0.5}, {0,0,-1}, {1,0,0}},
  {{3.86,0.93,1}, {0,0,-1}, {1,0,0}},
  {{3.86,0.93,0.6501}, {0,0,-1}, {1,0,0}} }, 3)
OpenGripper()
MoveThroughTo({ {{3.86,0.93,1}, {0,0,-1}, {1,0,0}},
  {{6.42,1,0.5}, {0,0,-1}, {1,0,0}},
  {{6.42,1,0.1501}, {0,0,-1}, {1,0,0}} }, 3)
CloseGripper()
MoveThroughTo({ {{6.42,1,0.5}, {0,0,-1}, {1,0,0}},
  {{4.14,0.93,1}, {0,0,-1}, {1,0,0}},
  {{4.14,0.93,0.6501}, {0,0,-1}, {1,0,0}} }, 3)
OpenGripper()
MoveThroughTo({ {{4.14,0.93,1}, {0,0,-1}, {1,0,0}},
  {{7.61,1.02,0.5}, {0,0,-1}, {1,0,0}},
  {{7.61,1.02,0.1501}, {0,0,-1}, {1,0,0}} }, 3)
CloseGripper()
MoveThroughTo({ {{7.61,1.02,0.5}, {0,0,-1}, {1,0,0}},
  {{4.14,1.07,1}, {0,0,-1}, {1,0,0}},
  {{4.14,1.07,0.6501}, {0,0,-1}, {1,0,0}} }, 3)
OpenGripper()
MoveThroughTo({ {{4.14,1.07,1}, {0,0,-1}, {1,0,0}},
  {{8.75,1,1}, {0,0,-1}, {1,0,0}},
  {{8.75,1,0.475}, {0,0,-1}, {1,0,0}} }, 3)
OpenToolChanger()
MoveThroughTo({ {{8.75,1,1}, {0,0,-1}, {1,0,0}},
  {{8.25,1,1}, {0,0,-1}, {1,0,0}},
  {{8.25,1,0.5}, {0,0,-1}, {1,0,0}} }, 3)
CloseToolChanger()
MoveThroughTo({ {{8.25,1,1}, {0,0,-1}, {1,0,0}},
  {{4,1,1}, {0,0,-1}, {1,0,0}},
  {{4,1,0.5001}, {0,0,-1}, {1,0,0}} }, 3)
CloseGripper()
MoveThroughTo({ {{4,1,1}, {0,0,-1}, {1,0,0}},
  {{2.5,1,1}, {0,0,-1}, {1,0,0}},
  {{2.5,1,0.0001}, {0,0,-1}, {1,0,0}} }, 3)
OpenGripper()
MoveThroughTo({ {{2.5,1,1}, {0,0,-1}, {1,0,0}},
  {{0.5,0,2}, {0,0,-1}, {1,0,0}} }, 2)
EndCanon(2)
```

Appendix B

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="urn:kitting"
 targetNamespace="urn:kitting"
 elementFormDefault="qualified"
 attributeFormDefault="unqualified"
 version="2013Jan10D">

 <xs:element name="KittingWorkstation"
  type="KittingWorkstationType">
 </xs:element>

 <xs:simpleType name="AngleUnitType">
  <xs:restriction base="xs:NMTOKEN">
   <xs:enumeration value="degree"/>
   <xs:enumeration value="radian"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:complexType name="BoxVolumeType">
  <xs:complexContent>
   <xs:extension base="DataThingType">
    <xs:sequence>
     <xs:element name="MaximumPoint"
      type="PointType"/>
     <xs:element name="MinimumPoint"
      type="PointType"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="BoxyShapeType">
  <xs:complexContent>
   <xs:extension base="InternalShapeType">
    <xs:sequence>
     <xs:element name="Length"
      type="PositiveDecimalType"/>
     <xs:element name="Width"
      type="PositiveDecimalType"/>
     <xs:element name="Height"
      type="PositiveDecimalType"/>
     <xs:element name="HasTop"
      type="xs:boolean"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="DataThingType"
  abstract="true">
   <xs:sequence>
    <xs:element name="Name"
     type="xs:ID"/>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="EndEffectorType"
  abstract="true">
   <xs:complexContent>
    <xs:extension base="SolidObjectType">
     <xs:sequence>
      <xs:element name="Description"
       type="xs:string"/>
      <xs:element name="Weight"
       type="PositiveDecimalType"/>
      <xs:element name="MaximumLoadWeight"
       type="PositiveDecimalType"/>
      <xs:element name="HeldObject"
       type="SolidObjectType"
       minOccurs="0"/>
     </xs:sequence>
    </xs:extension>
   </xs:complexContent>
 </xs:complexType>

 <xs:complexType
   name="EndEffectorChangingStationType">
   <xs:complexContent>
    <xs:extension base="SolidObjectType">
     <xs:sequence>
      <xs:element name="Base"
       type="MechanicalComponentType"/>
      <xs:element name="EndEffectorHolders"
       type="EndEffectorHolderType"
       maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:extension>
   </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="EndEffectorHolderType">
   <xs:complexContent>
    <xs:extension base="SolidObjectType">
     <xs:sequence>
      <xs:element name="EndEffector"
       type="EndEffectorType"
       minOccurs="0"/>
     </xs:sequence>
    </xs:extension>
   </xs:complexContent>
 </xs:complexType>
```

```
<xs:complexType name="ExternalShapeType">
  <xs:complexContent>
   <xs:extension base="ShapeDesignType">
    <xs:sequence>
     <xs:element name="ModelTypeName"
      type="xs:string"/>
     <xs:element name="ModelFileName"
      type="xs:string"/>
     <xs:element name="ModelName"
      type="xs:string"
      minOccurs="0"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="GripperEffectorType">
  <xs:complexContent>
   <xs:extension base="EndEffectorType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="InternalShapeType"
 abstract="true">
  <xs:complexContent>
   <xs:extension base="ShapeDesignType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="KitType">
  <xs:complexContent>
   <xs:extension base="SolidObjectType">
    <xs:sequence>
     <xs:element name="DesignName"
      type="xs:IDREF"/>
     <xs:element name="Tray"
      type="KitTrayType"/>
     <xs:element name="Part"
      type="PartType"
      minOccurs="0"
      maxOccurs="unbounded"/>
     <xs:element name="Finished"
      type="xs:boolean"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="KitDesignType">
 <xs:complexContent>
  <xs:extension base="DataThingType">
   <xs:sequence>
    <xs:element name="KitTraySkuName"
     type="xs:IDREF"/>
    <xs:element name="PartRefAndPose"
     type="PartRefAndPoseType"
     maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="KittingWorkstationType">
  <xs:complexContent>
   <xs:extension base="SolidObjectType">
    <xs:sequence>
     <xs:element name="AngleUnit"
      type="AngleUnitType"/>
     <xs:element name="ChangingStation"
      type="EndEffectorChangingStationType"/>
     <xs:element name="KitDesign"
      type="KitDesignType"
      maxOccurs="unbounded"/>
     <xs:element name="LengthUnit"
      type="LengthUnitType"/>
     <xs:element name="Object"
      type="SolidObjectType"
      maxOccurs="unbounded"/>
     <xs:element name="OtherObstacle"
      type="BoxVolumeType"
      minOccurs="0"
      maxOccurs="unbounded"/>
     <xs:element name="Robot"
      type="RobotType"/>
     <xs:element name="Sku"
      type="StockKeepingUnitType"
      maxOccurs="unbounded"/>
     <xs:element name="WeightUnit"
      type="WeightUnitType"/>
     <xs:element name="WorkTable"
      type="WorkTableType"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="KitTrayType">
  <xs:complexContent>
   <xs:extension base="SolidObjectType">
    <xs:sequence>
     <xs:element name="SkuName"
      type="xs:IDREF"/>
     <xs:element name="SerialNumber"
      type="xs:NMTOKEN"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType
 name="LargeBoxWithEmptyKitTraysType">
 <xs:complexContent>
  <xs:extension base="SolidObjectType">
   <xs:sequence>
    <xs:element name="LargeContainer"
     type="LargeContainerType"/>
    <xs:element name="KitTrays"
     type="KitTrayType"
     minOccurs="0"
     maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="LargeBoxWithKitsType">
 <xs:complexContent>
  <xs:extension base="SolidObjectType">
   <xs:sequence>
    <xs:element name="LargeContainer"
     type="LargeContainerType"/>
    <xs:element name="Kit"
     type="KitType"
     minOccurs="0"
     maxOccurs="unbounded"/>
    <xs:element name="KitDesignName"
     type="xs:IDREF"/>
    <xs:element name="Capacity"
     type="xs:positiveInteger"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="LargeContainerType">
 <xs:complexContent>
  <xs:extension base="SolidObjectType">
   <xs:sequence>
    <xs:element name="SkuName"
     type="xs:IDREF"/>
    <xs:element name="SerialNumber"
     type="xs:NMTOKEN"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:simpleType name="LengthUnitType">
 <xs:restriction base="xs:NMTOKEN">
  <xs:enumeration value="meter"/>
  <xs:enumeration value="millimeter"/>
  <xs:enumeration value="inch"/>
 </xs:restriction>
</xs:simpleType>

<xs:complexType
 name="MechanicalComponentType">
 <xs:complexContent>
  <xs:extension base="SolidObjectType"/>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="PartType">
 <xs:complexContent>
  <xs:extension base="SolidObjectType">
   <xs:sequence>
    <xs:element name="SkuName"
     type="xs:IDREF"/>
    <xs:element name="SerialNumber"
     type="xs:NMTOKEN"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="PartRefAndPoseType">
 <xs:complexContent>
  <xs:extension base="DataThingType">
   <xs:sequence>
    <xs:element name="SkuName"
     type="xs:IDREF"/>
    <xs:element name="Point"
     type="PointType"/>
    <xs:element name="XAxis"
     type="VectorType"/>
    <xs:element name="ZAxis"
     type="VectorType"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="PartsBinType">
 <xs:complexContent>
  <xs:extension base="SolidObjectType">
   <xs:sequence>
    <xs:element name="SkuName"
     type="xs:IDREF"/>
    <xs:element name="SerialNumber"
     type="xs:NMTOKEN"/>
    <xs:element name="PartSkuName"
     type="xs:IDREF"/>
    <xs:element name="PartQuantity"
     type="xs:nonNegativeInteger"/>
   </xs:sequence>
  </xs:extension>
```

```
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="PartsTrayType">
  <xs:complexContent>
   <xs:extension base="SolidObjectType">
    <xs:sequence>
     <xs:element name="SkuName"
      type="xs:IDREF"/>
     <xs:element name="SerialNumber"
      type="xs:NMTOKEN"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="PartsTrayWithPartsType">
  <xs:complexContent>
   <xs:extension base="SolidObjectType">
    <xs:sequence>
     <xs:element name="PartsTray"
      type="PartsTrayType"/>
     <xs:element name="Part"
      type="PartType"
      minOccurs="0"
      maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="PhysicalLocationType"
 abstract="true">
 <xs:complexContent>
  <xs:extension base="DataThingType">
   <xs:sequence>
    <xs:element name="RefObject"
     type="xs:IDREF"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="PointType">
  <xs:complexContent>
   <xs:extension base="DataThingType">
    <xs:sequence>
     <xs:element name="X"
      type="xs:decimal"/>
     <xs:element name="Y"
      type="xs:decimal"/>
     <xs:element name="Z"
      type="xs:decimal"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="PoseLocationType"
 abstract="true">
 <xs:complexContent>
  <xs:extension base="PhysicalLocationType">
   <xs:sequence>
    <xs:element name="Point"
     type="PointType"/>
    <xs:element name="XAxis"
     type="VectorType"/>
    <xs:element name="ZAxis"
     type="VectorType"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="PoseLocationInType">
 <xs:complexContent>
  <xs:extension base="PoseLocationType"/>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="PoseLocationOnType">
 <xs:complexContent>
  <xs:extension base="PoseLocationType"/>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="PoseOnlyLocationType">
 <xs:complexContent>
  <xs:extension base="PoseLocationType"/>
 </xs:complexContent>
</xs:complexType>

<xs:simpleType name="PositiveDecimalType">
 <xs:restriction base="xs:decimal">
  <xs:minExclusive value="0"/>
 </xs:restriction>
</xs:simpleType>

<xs:complexType name="RelativeLocationType"
 abstract="true">
 <xs:complexContent>
  <xs:extension base="PhysicalLocationType">
   <xs:sequence>
    <xs:element name="Description"
     type="xs:string"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
```

```
</xs:complexType>

<xs:complexType name="RelativeLocationInType">
  <xs:complexContent>
   <xs:extension base="RelativeLocationType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="RelativeLocationOnType">
  <xs:complexContent>
   <xs:extension base="RelativeLocationType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="RobotType">
  <xs:complexContent>
   <xs:extension base="SolidObjectType">
    <xs:sequence>
     <xs:element name="Description"
      type="xs:string"/>
     <xs:element name="EndEffector"
      type="EndEffectorType"
      minOccurs="0"/>
     <xs:element name="MaximumLoadWeight"
      type="PositiveDecimalType"/>
     <xs:element name="WorkVolume"
      type="BoxVolumeType"
      maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ShapeDesignType"
  abstract="true">
   <xs:complexContent>
    <xs:extension base="DataThingType">
     <xs:sequence>
      <xs:element name="Description"
       type="xs:string"/>
      <xs:element name="GraspPose"
       type="PoseLocationType"
       minOccurs="0"/>
     </xs:sequence>
    </xs:extension>
   </xs:complexContent>
</xs:complexType>

<xs:complexType name="SolidObjectType"
  abstract="true">
   <xs:sequence>
   <xs:element name="Name"
    type="xs:ID"/>
   <xs:element name="PrimaryLocation"
    type="PhysicalLocationType"/>
   <xs:element name="SecondaryLocation"
    type="PhysicalLocationType"
    minOccurs="0"
    maxOccurs="unbounded"/>
   <xs:element name="InternalShape"
    type="InternalShapeType"
    minOccurs="0"/>
   <xs:element name="ExternalShape"
    type="ExternalShapeType"
    minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="StockKeepingUnitType">
  <xs:complexContent>
   <xs:extension base="DataThingType">
    <xs:sequence>
     <xs:element name="Description"
      type="xs:string"/>
     <xs:element name="Shape"
      type="ShapeDesignType"/>
     <xs:element name="Weight"
      type="PositiveDecimalType"/>
     <xs:element name="EndEffectorName"
      type="xs:IDREF"
      minOccurs="0"
      maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="VacuumEffectorType"
  abstract="true">
   <xs:complexContent>
    <xs:extension base="EndEffectorType">
     <xs:sequence>
      <xs:element name="CupDiameter"
       type="PositiveDecimalType"/>
      <xs:element name="Length"
       type="PositiveDecimalType"/>
     </xs:sequence>
    </xs:extension>
   </xs:complexContent>
</xs:complexType>

<xs:complexType
  name="VacuumEffectorMultiCupType">
   <xs:complexContent>
    <xs:extension base="VacuumEffectorType">
     <xs:sequence>
      <xs:element name="ArrayNumber"
       type="xs:positiveInteger"/>
```

```
        <xs:element name="ArrayRadius"
          type="PositiveDecimalType"/>
       </xs:sequence>
     </xs:extension>
   </xs:complexContent>
 </xs:complexType>

 <xs:complexType
   name="VacuumEffectorSingleCupType">
  <xs:complexContent>
    <xs:extension base="VacuumEffectorType"/>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="VectorType">
   <xs:complexContent>
    <xs:extension base="DataThingType">
      <xs:sequence>
        <xs:element name="I"
          type="xs:decimal"/>
        <xs:element name="J"
          type="xs:decimal"/>
        <xs:element name="K"
          type="xs:decimal"/>
       </xs:sequence>
     </xs:extension>
   </xs:complexContent>
 </xs:complexType>

 <xs:simpleType name="WeightUnitType">
   <xs:restriction base="xs:NMTOKEN">
     <xs:enumeration value="kilogram"/>
     <xs:enumeration value="gram"/>
     <xs:enumeration value="milligram"/>
     <xs:enumeration value="ounce"/>
     <xs:enumeration value="pound"/>
   </xs:restriction>
 </xs:simpleType>

 <xs:complexType name="WorkTableType">
   <xs:complexContent>
    <xs:extension base="SolidObjectType">
      <xs:sequence>
        <xs:element name="SolidObject"
          type="SolidObjectType"
          minOccurs="0"
          maxOccurs="unbounded"/>
       </xs:sequence>
     </xs:extension>
   </xs:complexContent>
 </xs:complexType>

</xs:schema>
```

Appendix C XML Schema File scoreKitting.xsd

```
<?xml version="1.0"?>
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="urn:Kitting"
 targetNamespace="urn:Kitting"
 attributeFormDefault="unqualified"
 elementFormDefault="qualified">

 <xs:element name="scoreKitting"
  type="xsi:scoreKittingType">
 </xs:element>

 <xs:complexType name="scoreKittingType">
  <xs:sequence>
   <xs:element name="rightStuff"
    type = "xsi:factorValueOptType">
   </xs:element>
   <xs:element name="commandExecution"
    type = "xsi:factorValueOptType">
   </xs:element>
   <xs:element name="distance"
    type = "xsi:factorValueOptType">
   </xs:element>
   <xs:element name="time"
    type = "xsi:factorValueOptType">
   </xs:element>
   <xs:element name="uselessCommands"
    type = "xsi:factorValueReqType">
   </xs:element>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="factorType"
  abstract="true">
   <xs:sequence>
    <xs:element name="isAdditive"
     type="xs:boolean">
    </xs:element>
    <xs:element name="weight"
     type="xs:unsignedInt">
    </xs:element>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="factorValueOptType">
   <xs:complexContent>
    <xs:extension base="xsi:factorType">
     <xs:sequence>
      <xs:element name="valueFunction"
       type="xsi:valueFunctionType"
        minOccurs="0">
       </xs:element>
     </xs:sequence>
    </xs:extension>
   </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="factorValueReqType">
  <xs:complexContent>
   <xs:extension base="xsi:factorType">
    <xs:sequence>
     <xs:element name="valueFunction"
      type = "xsi:valueFunctionType">
      </xs:element>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="valueFunctionType">
  <xs:sequence>
   <xs:element name="bestValue"
    type = "xs:double">
   </xs:element>
   <xs:element name="width"
    type = "xsi:nonNegativeReal">
   </xs:element>
   <xs:element name="taper"
    type = "xsi:nonNegativeReal">
   </xs:element>
   <xs:element name="taperSide"
    type = "xsi:taperSideType">
   </xs:element>
  </xs:sequence>
 </xs:complexType>

 <xs:simpleType name="nonNegativeReal">
  <xs:restriction base="xs:double">
   <xs:minInclusive value="0.0"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="taperSideType">
   </xs:annotation>
  <xs:restriction base="xs:string">
   <xs:enumeration value="plus"/>
   <xs:enumeration value="minus"/>
   <xs:enumeration value="both"/>
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```