

AUTOMATICALLY RESOLVING BOUNDARIES WITHIN AN ARCHITECTURAL DESIGN ENVIRONMENT

BACKGROUND OF THE INVENTION

5 The Field of the Invention

The present invention relates generally to computer-aided design or drafting software.

Background and Relevant Technology

10 As computerized systems have increased in popularity, so have the range of applications that incorporate computational technology. Computational technology now extends across a broad range of applications, including a wide range of productivity and entertainment software. Indeed, computational technology and related software can now be found in a wide range of generic applications that are suited for many environments, as well as fairly industry-specific software.

15 One such industry that has employed specific types of software and other computational technology increasingly over the past few years is that related to building and/or architectural design. In particular, architects and interior designers (“or designers”) use a wide range of computer-aided design (CAD) software for designing the aesthetic as well as functional aspects of a given residential or commercial space. For example, a designer might use a CAD program to design
20 fixtures and furniture for a particular office. The designer might then export the designs to be manufactured by a particular millwork facility.

While millwork is becoming a more common method of producing furniture, producing custom millwork furniture can be an expensive and time-consuming
25 process. For example, conventional systems may require that custom furniture first be meticulously designed within a CAD program. Additionally, prior to designing the furniture or fixture within the CAD program, conventional systems may require that the specifications of the end product be exactly known ahead of time. For instance, a designer may need to know the exact dimensions of the object being designed, along
30 with the finishing features, such as joint type, hinge type, door sizes, etc.

Additionally, in at least some conventional systems, a designer or manufacturer may incur great expense if any design changes are made after the initial

CAD model is created. If, for example, a designer discovers that an initial measurement was incorrect, the designer may need to adjust or even recreate the entire design manually, taking into account the correct measurement. In addition, large cost can be incurred by simply switching from one millwork provider to another.

- 5 For example, different millwork providers may use different joints, different hardware, different materials, materials of different dimensions, etc. As mentioned above, even slight changes such as these may require significant reworking of the CAD design.

Accordingly, there are a number of problems in the art relating to modeling
10 architectural elements within a CAD program and later manufacturing those elements with a millwork facility.

BRIEF SUMMARY OF THE INVENTION

Implementations of the present invention overcome one or more problems in the art with systems, methods, and computer program products for automatically
15 resolving anomalies on the boundaries of an architectural design. For example, in at least one implementation of the present invention, the millwork software can automatically identify an anomaly and determine one or more options for correcting the anomaly. The millwork software can then automatically resolve the anomaly or provide a user with one or more options for resolving the anomaly. One will
20 appreciate that this can provide a number of advantages to designers incorporating particular finish details, such as millwork, into complex designs.

For example, a method in accordance with at least one implementation for automatically resolving joint anomalies within digital joints of an architectural design can include receiving a digital architectural design comprising a first furniture sub-
25 component, a second furniture sub-component, and a third furniture sub-component. The method can then identify one or more joints between the various furniture sub-components. After identifying the joints, the method can include identifying an anomaly at the intersection of the joints. For example, the anomaly can be created when the joints fail to create a proper corner. The method can then automatically
30 resolve the anomaly by changing the type of at least one of the joints within the digital architectural design.

In an additional or alternative implementation, a method can include analyzing a digital architectural design comprising one or more furniture sub-components. Each of the furniture sub-components can be associated with one or more independently executable software objects. The method can then include identifying various joints
5 created by the one or more sub-components. After identifying the joints, the method can include identifying an anomaly at the intersection of the joints. For example, the anomaly can be created when the joints fail to create a proper corner. The method can further comprise accessing information provided by the one or more independently executable software objects. The accessed information can comprise one or more
10 characteristics of the architectural design. Based upon the accessed one or more characteristics, the method can further involve determining one or more joint specifications that are compatible with the architectural design. Upon identifying compatible joints, the method can include automatically resolving the anomaly by changing the type of at least one of the joints to comprise a revised joint that conforms
15 with the one or more joint specifications.

These and other objects and features of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

20 In order to describe the manner in which the above-recited and other advantages and features of the invention can be obtained, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments thereof, which are illustrated in the appended drawings. It should be noted that the figures are not drawn to scale, and that elements of similar
25 structure or function are generally represented by like reference numerals for illustrative purposes throughout the figures. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered to be limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

30 Figure 1 is a schematic illustration of a system for designing and manufacturing an architectural element in accordance with an implementation of the present invention;

Figures 2A illustrates a spatial framework created in accordance with the inventive system illustrated in Figure 1;

Figure 2B illustrates the spatial framework of Figure 2A, after a designer has inserted a cube splitter in accordance with an implementation of the present invention;

5 Figure 2C depicts the spatial framework of Figure 2B upon implementation of one or more facet splitters in accordance with an implementation of the present invention;

Figure 3 depicts a planar view of a spatial framework created in accordance with an implementation of the present invention;

10 Figure 4A depicts a spatial framework created in accordance with the present invention in which there are no joint anomalies;

Figure 4B depicts a similar spatial framework as that of Figure 4A, albeit with joint anomalies;

15 Figures 5A depicts an architectural design for a drawer in accordance with an implementation of the present invention;

Figure 5B depicts a front face of the drawer depicted in Figure 5A in accordance with an implementation of the present invention;

Figure 5C depicts a drawer bottom and side, albeit without a connecting joint detail in accordance with an implementation of the present invention;

20 Figure 5D depicts the drawer bottom and side of Figure 5C, now with a joint detail, in accordance with an implementation of the present invention;

Figure 5E depicts a storage unit and an oversized sink in accordance with an implementation of the present invention;

25 Figure 5F depicts a resized storage unit and an inset sink as depicted in Figure 5E in accordance with an implementation of the present invention;

Figure 6 depicts a finished rendering of an architectural element created in accordance with an implementation of the present invention;

Figure 7A illustrates a spatial framework created in accordance with the inventive system illustrated in Figure 1;

30 Figure 7B illustrates a second spatial framework embedded within the spatial framework of Figure 7A, created in accordance an implementation of the present invention;

Figure 8A depicts an oversized cabinet system and an alcove in accordance with an implementation of the present invention;

Figure 8B depicts the cabinet system of Figure 8A automatically adjusted to fit within the alcove in accordance with an implementation of the present invention;

5 Figure 9A depicts an undersized cabinet system and an alcove in accordance with an implementation of the present invention;

Figure 9B depicts the cabinet system of Figure 9A automatically adjusted to fit within the alcove in accordance with an implementation of the present invention;

10 Figure 10 illustrates a flowchart of a series of acts in a method in accordance with an implementation of the present invention for incorporating third party features into an architectural design; and

Figure 11 illustrates another flowchart of a series of acts in a method in accordance with an implementation of the present invention for incorporating third party features into an architectural design.

15 DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Implementations of the present invention extend to systems, methods, and computer program products for automatically resolving anomalies on the boundaries of an architectural design. For example, in at least one implementation of the present invention, the millwork software can automatically identify an anomaly and
20 determine one or more options for correcting the anomaly. The millwork software can then automatically resolve the anomaly or provide a user with one or more options for resolving the anomaly. One will appreciate that this can provide a number of advantages to designers incorporating particular finish details, such as millwork, into complex designs.

25 For example, in at least one implementation of the present invention, a user can use an object oriented CAD program to create a spatial framework representative of an architectural element, for example, a desk with drawers. As the user is creating the framework of the desk with drawers, the CAD program can automatically create the surfaces and spaces that will make up the architectural element by analyzing the
30 user's input. Additionally, the CAD program can identify potential uses for the spaces.

In at least one implementation, the one or more modules in the CAD program can identify and track space by assigning an independently executable software object to the space. As an independently executable software object, the space can have independent functions and variables associated with it. As needed, these functions
5 and variables can automatically adjust the space, and in turn the planes that define the space in response to manufacturing specifications and manufacturer specific components.

Additionally, in at least one implementation, the present invention can automatically incorporate specific features and specifications from a third party
10 manufacturer. In many cases, different manufacturers may use different hardware, different types of joints, and various other different manufacturing elements that each may require unique adjustments to an architectural design. For example, when manufacturing a kitchen, a first millwork facility may use a smaller sink with only a single basin, while a second millwork facility may use a larger sink with two basins.
15 In at least one implementation, the architectural design can automatically adjust to incorporate either the smaller sink or the larger sink. One will appreciate that the ability to automatically incorporate third party features into an architectural design after the design is substantially complete can provide significant benefits.

Additionally, in at least one implementation, the present invention can aid in
20 automatically resolving anomalies created. In at least one implementation, for example, one or more components of a CAD program can incorporate manufacturer specific components and specifications into the architectural design. For example, a particular manufacturer may use a specific type of carpentry joint. Some combinations of joints, however, may result in an impossible configuration or a
25 configuration with an anomaly – often at a corner. In at least one implementation, the present invention can automatically identify an impossible or invalid configuration or a configuration that may create an anomaly, and then automatically resolve the conflict to create an architectural element with appropriate joints.

Implementations of the present invention can also allow a user to create a
30 framework for an architectural element without knowing many of the end features that will be incorporated into the architectural element. For example, in at least one implementation, a user can create a framework for an entire kitchen without knowing

the material that will be used, the dimensions of the resulting kitchen, the specific fixtures, and many other similar details. Once the digital framework is completed, a specific millwork facility can enter in the details of its materials, fixtures, configurations, and the actual specifications and/or dimensions of the end product;
5 and the digital framework can automatically adjust to conform to the entered information.

Turning now to the Figures, Figure 1 depicts an architectural schematic diagram of a computer system for designing and manufacturing an architectural element. In particular, Figure 1 shows that an exemplary computer terminal 110 can
10 be in communication with a millwork software application 100. The millwork software application 100 can be executed from the computer terminal 110, from a server (not shown) that the computer terminal 110 is accessing, or by using some other known method of execution. One will appreciate that the computer terminal 110 can comprise any computerized device that enables execution of computerized
15 instructions, including any desktop, laptop, or mobile computing device.

Figure 1 further shows that the millwork software application 100 can comprise a plurality of modules 120, 130, 140, 142, 144, 146, 148, 160 that are adapted to aid in designing a file for millwork. Each module 120, 130, 140, 142, 144, 146, 148, 160 may be representative of various programmatic functionality. For
20 example, Figure 1 depicts an implementation in which the millwork software application 100 comprises a user interface module 120, a manufacturing preparation module 130, a framework module 140, a facet module 142, a spaces module 144, a boundary module 146, a recursion module 148, a configuration list module 160, and a storage device 150. One will understand, however, that separating modules into
25 discrete units is at least somewhat arbitrary and that modules can be combined, associated, or separated in ways other than shown in Figure 1 and still accomplish the purposes of this invention. Accordingly, the particular modules 120, 130, 140, 142, 144, 146, 148, 160 of Figure 1 are only shown for illustrative and exemplary purposes of at least one implementation.

30 In addition, Figure 1 shows that the user interface module 120 can communicate with the computer terminal 110 through a series of data packets 112. For example, the user interface module 120 can display images and graphical controls

to a user through a computer monitor and can receive input from a user through a keyboard and/or mouse. As a user creates and/or manipulates a particular framework of an architectural element, the user interface module 120 can communicate to – and receive instructions from – the framework module 140. Figure 1 further shows that
5 the framework module 140 can, in turn, communicate with the facet module 142, the spaces module 144, the boundary module 146, the recursion module 148, and the configuration list module 160.

Ultimately, either user interface module 120 or framework module 140 can communicate with manufacturing preparation module 130 to create a file that is
10 prepared for use in a millwork facility. Additionally, Figure 1 shows that the various modules can communicate with a storage device 150. The storage device 150 can contain, among other things, templates for a variety of different designs, completed designs that can be used on a standalone basis, or that incorporated into other designs. One will appreciate that the storage device 150 can also contain tool lists and/or
15 manufacturing information specific to particular millwork facilities, and/or to particular design features.

One will appreciate in view of the specification and claims herein that the user interface module 120 provides to the user an option to create and make design changes to a framework 200. In at least one implementation for example, upon
20 receiving a request for some modification, the user interface module 120 can communicate the request to the framework module 140. For example, a user may desire to design a desk for production at a millwork facility. Accordingly, a user may enter instructions into the computer terminal 110 to design and create the desk. The user interface module 120 can, in turn, communicate those instructions to the
25 framework module 140.

Upon receiving the instructions, the framework module 140 can then communicate with the appropriate module to execute the request. For example, if the user desires to split the upper surface of the desk into two portions, the framework module 140 communicates with the facet module 142, which can modify and track
30 surfaces within the framework 200. In contrast, and as will be understood more fully herein, if the user desires to split a space into two spaces, the framework module 140

can alternatively use the spaces module 144, which can modify and track spaces within the framework 200.

For instance, the spaces module 144 can allow the user to split the framework 200 in half and create one half of the desk that is dedicated to drawers, and another
5 half that is open space for the user to place his or her chair and feet. Additionally, a user may use the spaces module 144 to split the framework 200 into any number of other divisions, for example thirds. Of course, one will appreciate that the divisions do not need to be proportionally equal. For instance, the spaces module 144 can allow a user to move the single split mentioned above such that the drawers of the
10 desk only take up one-third of the framework, while the leg space takes up the remaining two-thirds.

In at least one implementation, the facet module 142 can perform similar functions on surfaces within the framework 200. For example, the facet module 142 can be used to create a split in the front surface of the desk to create doors for a
15 cupboard. Similar to the spaces module 144, the facet module 142 can also be used to create split surfaces of different sizes.

In addition, the framework module 140 can use the boundary module 146 to automatically check joints within the framework 200 (Figures 2A-2B) to determine if any “anomalies” exist. For example, if a user specifies that particular joint should be
20 a miter joint, the boundary module 146 can analyze all of the joints within the desk to determine whether the remaining boundaries properly form around the entire desk. If the boundary module 146 detects any anomalies, (e.g., improperly overlapped joints, or other inappropriate positioning), the boundary module 146 can automatically resolve them and create proper joints throughout the desk.

25 Once the user has finished designing the architectural element (in this case a desk), the manufacturing preparation module 130 can receive millwork facility specific details from the configuration list module 160, and can receive specifications relating to the final details of the architectural element. For example, a particular millwork facility may use a dovetail joint to assemble the drawers of the dresser.
30 Additionally, the millwork facility may use a particular type of wood that comprises a specific thickness. Upon receiving this information from the configuration list module 160, the manufacturing preparation module 130 can automatically adjust the

framework 200 of the desk to create a design that incorporates the dovetail joint, wood type and wood thickness, and that can be manufactured at the millwork facility. In contrast, in at least one implementation, the manufacturing preparation module 130 may not make any changes to the actual framework 200, but instead the manufacturing preparation module 130 may make the necessary changes to the actual manufacturing code (e.g., CNC code).

Similarly, the manufacturing preparation module 130 can automatically adjust the framework 200 of the desk to incorporate specific third party features. For example, a designer may originally design a desk without knowing the specific handles that a particular millwork facility uses for the drawers. In at least one implementation, the configuration list module 160 can receive a list of components and specifications that the particular millwork facility uses. The manufacturing preparation module 130 can then automatically incorporate those components and specifications of the millwork facility into the design.

For example, the millwork facility may use handles that require two pre-drilled screw holes spaced 10 cm apart with the midpoint between the holes being centered on the drawer face. Upon receiving this information from the configuration list module 160, the manufacturing prepared module 130 can automatically incorporate the two screw holes for the handle into the desk drawer. Additionally, the manufacturing preparation module 130 can automatically incorporate the holes into a computer numerical code ("CNC") file that the particular millwork facility can use to automate the manufacture of the desk.

In at least one implementation, the manufacturing preparation module 130 can also automatically prepare the same framework 200 to be manufactured in any number of different millwork facilities, even though each facility may have specific manufacturing requirements. Specifically, once a particular framework 200 has been designed, the framework 200 can be used at a number of different millwork facilities as long as the configuration list module 160 has access to a configuration list 152 that is associated with the specific facility. The configuration list 152 can be stored in the storage device 150, accessed through a network connection, or otherwise made available to the millwork software 100.

Figures 2A-2C depict various implementations of a spatial framework 200. As shown in Figure 2A, the spatial framework 200 can initially comprise a simple cube. In at least one implementation, the millwork software 100 can comprise a plurality of different simply shaped frameworks 200 that can be used as starting
5 points for designing an architectural element. In general, the spatial framework 200 can be a computer model of an architectural element that captures the design intent of a user.

In particular, the spatial framework 200 can capture data relating to the outline of an architectural element and the position of components in the element with respect
10 to each other. To accomplish this, the spatial framework 200 can comprise a space 250 that is associated with an independently executable software object. The independently executable software object can assist in tracking and managing the various components of the designed architectural element. In at least one implementation, the space 250 and independently executable software object are both
15 managed by the spaces module 144.

By way of explanation, an independently executable software object comprises a set of computer-executable instructions used in object-oriented program code, and which relate to a particular physical component or feature. In addition, software objects can be interrelated via parent/child dependency relationships where
20 changes in a parent object flow through to a child object and vice versa. For example, a software object created for a table may have several child objects for each leg.

In other cases, the software objects can be related to other software objects that represent physically proximate components (e.g., a wall object that is positioned next to the table object). For example the above-mentioned table software object and
25 leg software objects can independently execute in a correlated fashion to ensure each corresponding physical component (i.e., the table top, or the table legs) is positioned appropriately, or otherwise colored and designed consistent with the user's specifications. For example, a leg software object can identify that it's location next to a wall renders a physical leg unnecessary, and accordingly can automatically
30 incorporate a bracket to attach directly to the wall in place of a physical leg.

Figure 2A also depicts that the framework 200 can comprise "facets" 202, 204, 206 and boundaries 208. By way of explanation, "facets" 202, 204, 206

represent surfaces within the framework. Facets 202, 204, 206, however, may not always correlate to surfaces within the finished architectural element. For example, facets 202, 204, 206 may only be quasi-two-dimensional because they can comprise a specified thickness. In some cases, a user can set the thickness of a particular facet to be zero. As a result the facet can still be a part of the framework 200 but it will not be a part of the finished architectural element. “Boundaries,” on the other hand, represent lines where facets meet. In at least one implementation, a specific boundary’s location can be defined with respect to the other boundaries that the specific boundary intersects.

Figure 2B depicts an implementation of a framework 200 that has been bisected by a “cube splitter” 210. As depicted, the cube splitter 210 splits or divides space 250 in half creating two new spaces 212 and 214, which spaces may or may not comprise the same dimensions. In at least one implementation, the creation of two new spaces 212, 214 also results in the creation of two new independently executable software objects (no shown), which are associated with each space 212, 214. It should be understood that while cubes and squares are used to illustrate embodiments of the present invention within this application, in at least one implementation, many different shapes and configurations of a framework 200 can be used, and it should further be understood that a cube splitter can divide spaces into non-equal portions.

In particular, in at least one implementation, the newly created independently executable software objects associated with spaces 212 and 214 may each inherit the parameters and characteristics of the independently executable software object that was originally associated with space 250. In at least one implementation, due to this inheritance, if space 250 originally comprises a set of drawers, after the split, spaces 212 and 214 can each comprise separate, independently executable software objects that correspondingly comprise a set of drawers that mirror the original drawers of space 250.

The cube splitter 210 can also create a new facet 216 within the framework 200 and a plurality of new facets (for example 220, 222) on each external surface of the framework 200. As mentioned above, the new facets 216, 220, 222 can each comprise a unique thickness, such that the facets 216, 220, 222 comprise physical surfaces within the architectural element. Similarly, or alternatively, the facets 216,

220, 222 can comprise thicknesses of zero, resulting in the facets 216, 220, 222 only being represented within the framework 200 but not within the finished architectural element.

Figure 2C depicts the framework 200 of Figure 2B, albeit comprising two
5 facet splitters 230, 232. As depicted, the facet splitters 230, 232 can split facet 222 (created from splitting facet 204) into three new facets 234, 236, 238. In at least one implementation, a facet splitter 230, 232 is configured to split only facets, such as facets 220, 222, 234, 236, 238. This is as opposed to a cube splitter 210, which can be configured in at least one implementation to split an entire space 250 (i.e., into spaces
10 212, 214, Figures 2B-2C). In addition, in at least one implementation, the new facets 234, 236, 238 remain associated with space 214, and thus can be associated with the independently executable software object that is associated with space 214.

Figure 3 depicts a quasi-two-dimensional view of front face 204 of the framework 200 from Figure 2C, after various space and facet splitting. As mentioned
15 previously, the view is quasi-two-dimensional because each facet can in fact comprise a thickness, when so set by the user. Accordingly, Figure 3 shows facet 220, which was formed by the placement of cube splitter 210, and facets 234, 236, and 238, which were formed by the placement of facet splitters 230 and 232. Additionally, Figure 3 depicts the end point 300, 301, 302, 303, 304, 305, 306, 307, 320, 322 of
20 each boundary within the front face of the framework. In at least one implementation, a user can interact with the framework 200 through either a three-dimensional view (e.g., Figs. 2A-2C) or through a quasi-two-dimensional view (e.g., Fig. 3). In either view, the location and behavior of the facets can be managed by the facet module 142.

In at least one implementation, the location and positioning of the cube
25 splitters 210 and facet splitters 230, 236, 238 within the framework can be tracked with respect to the end points 300, 301, 302, 303, 304, 305, 306, 307, 320, 322 of each respective facet splitter 230, 236, 238 or cube splitter 210, and in particular, where those end points intersect other boundaries. In at least one implementation, the location of the end point intersections can be tracked as a finite distance or as a
30 proportion of the total length of the respective boundary.

For example, Figure 3 shows that facet splitter 230 comprises end points 301 and 305. In at least one implementation, the location of facet splitter 230 can be

designated as end point 301 being located distance 310 from the top of cube splitter 210, and end point 305 being located distance 312 from the top of side boundary 340. Similarly, the location of facet splitter 232 can be designated as end point 302 being positioned 1/3 up the length of cube splitter 210, and similarly, end point 306 being
5 location 1/3 up the length of side boundary 340.

One will understand that using a finite length or a proportional length can impact the future millwork of the architectural element. For example, if a user resizes an item framework 200 by expanding the framework in all directions, then facet splitter 230 can still be located distance 310 and 312 from the top of cube splitter 210,
10 and from the side boundary 340 respectively. Facet splitter 232, on the other hand, can change in absolute position such that each end point 302, 306 is 1/3 up the length of their respective boundaries 210, 340 (e.g., between points 300/303 and/or 304/307), without regard to the actual length of boundaries 210 and 340.

In at least one implementation, the framework 200 can be shrunk so much that
15 absolute distance 310 and distance 312 exceed 2/3 of the total length of boundaries 210 and 340 respectively. One will understand that this can cause facet splitter 230 to overlap facet splitter 232. In this situation, the framework module 140 can automatically determine that either facet splitter 232 or facet splitter 230 should automatically be removed leaving only a single facet splitter 230 or 232, but not both.
20 For example, in at least one implementation, a user can set an option to automatically give fixed lengths 310, 312 priority over proportional lengths 303, 316, or to automatically give proportional lengths 303, 316 priority over fixed lengths 310, 312.

Additionally, a user may be able to set an option that gives priority to the first facet splitter 230, 232, or to cube splitter 210 created over subsequent facet splitters
25 230, 232, or to cube splitter 210. In contrast, a user may be able to set an option that gives priority to the last facet splitter 230, 232, or to cube splitter 210 created over previous facet splitters 230, 232, or to cube splitter 210. Further, in at least one implementation, a user can specifically designate that a particular facet splitter 230, 232, or cube splitter 210, should be given priority over other facet splitters 230, 232,
30 and/or cube splitters 210.

Allowing a user to determine whether a facet splitter or cube splitter should be located on a proportional distance basis, or on an absolute distance basis, can provide

the user with significant control over how an architectural element can be resized and manipulated. Additionally, allowing a user to determine the priority that particular cube splitters and/or facet splitters can be given when the splitters conflict with each other allows a user to have control over the final configuration of an architectural
5 element that has been resized.

Figures 4A and 4B depict various implementations of boundary detection and correction. In at least one implementation, as a user designs a framework 200 for a particular architectural element, the user is able to specify the type of joints that the user desires to join particular surfaces of the architectural element. In contrast, in at
10 least one implementation, a particular millwork facility may use a specific joint type in some architectural elements. As such, a user may be able to adjust or specify select joints, while others are fixed to thereby accommodate the specifications of the millwork facility. Thus, as the user creates an architectural design, the boundary module 146 can analyze the design to verify that no anomalies exists within the
15 various joints, and that the joints are consistent with the given facility's specifications.

By way of explanation, "anomalies" can be created when non-compatible joint types meet at an intersection. In at least one particular implementation, an anomaly includes a portion of an architectural design that fails to form a proper corner. As will be explained further below, in some implementations, certain joint combinations will
20 create a gap in a location that should normally comprise a finished corner. Incompatible joints may also create anomalies that do not exist solely as incorrectly formed corners. For example, an anomaly can also comprise a corner that is not symmetric with at least one other corner of an architectural design of a piece of furniture. In this case, the corner is an anomaly because it does not match with the
25 rest of the design.

As an example, Figure 4A depicts a simple architectural element (e.g., a box) that comprises an upper surface 400(a) (or upper "sub-component"), a side surface 404(a) (or side "sub-component"), and a front surface 402(a) (or front "sub-component"). As depicted, the front surface 402(a) and the side surface 404(a) meet
30 each other at miter joint 420(a). Figure 4A further shows that the side surface 404(a) and the upper surface 400(a) meet each other at joint 430(a) where upper surface 400(a) overlaps side surface 404(a), and upper surface 400(a) and front surface 402(a)

meet each other at joint 410(a) where upper surface 400(a) overlaps front surface 402(a). In at least one implementation, the boundary module 146 can analyze the joints of Figure 4A and determine that the above-recited joints do not create any anomalies, and can thus be left as they are.

5 In contrast to Figure 4A, however, Figure 4B illustrates an alternative illustration of the same architectural element shown in Figure 4A, albeit one in which the joints do create an anomaly (440). In particular, Figure 4B shows an alternate arrangement of surfaces in which top surface 400(b) is placed on an inside surface of surface 402(b), and hence meets front surface 402(b) at joint 412 where front panel
10 402(b) overlaps the front edge of upper surface 400(b).

Additionally, the front surface 402(b) meets side surface 404(b) at joint 422 where side surface 404(b) overlaps the side edge of front surface 402(b). Further, Figure 4B shows that the side surface 404(b) meets upper surface 400(b) at joint 432 where upper surface 400(b) overlaps side surface 404(b). As shown,, this particular
15 selection of joints creates an anomaly 440.

In at least one implementation, the boundary module 146 can analyze the joints of Figure 4B and identify the presence of an anomaly 440. For example, the boundary module 146 can access a database that contains a list of joint configurations that have been predetermined to create an anomaly, when these surfaces 400(a & b),
20 402(a & b), and 404(a & b) are arranged in this alternate configuration. If the boundary module 146 detects that the arrangement of Figure 4B is already listed as an anomaly within the database list in storage device 150, the boundary module 146 can then immediately determine that an anomaly exists without any additional calculations or analysis.

25 In another implementation, the boundary module 146 can analyze each of the joints within the architectural design, and based upon the analysis identify that an anomaly exists. For example, the boundary module 146 can determine that a particular configuration of joints does not create a correct corner where each of the joints meets, based on the arrangement of surfaces 400(b) and 402(b) relative to the
30 edges of surface 402(b). As previously discussed with respect to Figure 4B, for example, instead of creating a correct corner, an empty gap (i.e., anomaly 440) is present where the corner should exist. The boundary module 146 can be adaptive to

similarly identify anomalies in situations where more than three surfaces are meeting at a point, or when multiple surfaces meet at angles that do not necessarily create a corner. Further, in at least one implementation, the boundary module 146 can identify an anomaly by identifying a corner that is not symmetric with at least one other corner
5 in a particular design.

Once an anomaly (e.g., 440) is identified, the boundary module 146 can resolve the anomaly by automatically adjusting the joints. For example, in Figure 4B, the boundary module 146 can change boundary 422 such that the front panel 402(a & b) is rearranged to overlap the side panel 404(a & b) (as in Figure 4A), and by
10 correspondingly rearranging surfaces 400(a & b) and/or 402(a & b). One will appreciate that this change in the joint configuration will resolve the anomaly. In at least one implementation, after identifying a correction to the anomaly, the user interface module 120 will display to the designer a depiction of architectural design, including the corrected joint, and provide a prompt for the user to accept or reject the
15 change. If the user rejects the change, the boundary module 146 can re-analyze the architectural design, and present one or more additional correct options for the user to choose between.

In determining what boundaries to change, the boundary module 146 can operate such that the joints specified by a configuration list from a millwork facility
20 are preserved while other joints are changed. In contrast, the boundary module 146 can adjust the joints such that the most recently specified joint is preserved and others are changed. Alternatively, the boundary module 146 can change joints such that the earliest specified joints are preserved and the most recently specified joints are changed. In addition, in at least one implementation, a user can specify that a
25 particular joint be given priority over other joints. For example, upon detecting an anomaly (e.g., 440), the user interface module 120 can prompt a user to select a preferred joint that will be preserved by the boundary module 146. Accordingly, one or more implementations present a variety of different procedures for determining which joints to preserve and which joints to change.

30 As an additional example of a method for determining which joints to change, the boundary module 146 can automatically analyze the architectural design and determine a preferred joint type based upon other joints present throughout the

architectural design. In analyzing the architectural design, the boundary module 146 can analyze joints associated with multiple distinct components within the design, joints adjacent to the anomaly, joints on the same face as the anomaly, or joints otherwise present within an architectural design.

5 For example, a particular architectural design can comprise a desk, a shelf, and a filing cabinet. The boundary module 146 may determine that the desk, for example, comprises an anomaly (e.g., 440) created by an improper joint configuration. In at least one implementation, the boundary module 146 can then automatically analyze the joints that are used within the desk, the shelf, and/or the filing cabinet to
10 determine if a particular joint type is more common. Similarly, the boundary module 146 can analyze the desk to determine the type of joints that are adjacent to the joint that is being changed, and identify the adjacent joint types as being preferred. Additionally, if the anomaly is along the writing surface of the desk, the boundary module 146 can analyze the other joints that surround the writing surface to identify a
15 preferred joint.

 One will appreciate that this can provide a visual benefit by keeping joint placement consistent. For example, it may be more visually appealing for all of the joints along the top of the desk to be of the same type. Once a preferred joint type is identified (whether by the user, or based on an automatic configuration), the boundary
20 module 146 can correct the anomaly within the desk while preserving any joints in the desk that are of the preferred type, or by changing one or more joints to match the preferred type.

 Additionally, in at least one implementation, when correcting anomalies within the joints of an architectural design, the boundary module 146 can
25 communicate with one or more independently executable software objects that are associated with the surfaces 400, 402, 404 of the design, the joints 410(a & b), 420(a & b), 430(a & b) of the design, and/or with independently executable software objects that are otherwise associated with the design. The independently executable software objects can provide information relating to joints that can be incorporated into the
30 design and/or joints that cannot be incorporated into the design.

 The independently executable software objects, in turn, may also indicate preferred joint configurations that can be used to resolve anomalies. Additionally, the

independently executable software objects can provide information such as available materials, dimensions, strength requirements, and other related characteristics that can be used to determine which joints should be used. For example, if a stone, such as granite or marble, is being used as a material, the user may desire to use an overlap joint to better display the stone grain. Accordingly, in the case, an independently executable software object can indicate that the material is stone; and, as a result, the boundary module 146 can use this information to determine that an overlap joint is preferred.

In at least one implementation, the boundary module 146 can allow a millwork facility to specify particular types of joints for constructing an architectural element. A designer, therefore, can design the complete architectural design without being aware of the particular specifications of the millwork facility that will manufacture an architectural element. Additionally, a designer can create a single architectural design that the manufacturing preparation module 130 can automatically adjust, as dictated by a millwork facility specific configuration list provided to the configuration list module 160. In this way a single architectural design can be used by a variety of different millwork facilities, each of which requires unique joints and specifications.

Once an architectural element has been sufficiently designed within a framework 200, the manufacturing preparation module 130 can prepare the framework 200 for actual production at a millwork facility. For example, the configuration list module 160 can provide the manufacturing preparation module 130 with the various specifications, hardware components, and other manufacturing constraints of a particular millwork facility. In at least one implementation, this can include adjusting the framework 200 to incorporate a specific material type, or a specific material thickness, adjusting the framework 200 to fit within a particular space, incorporating the appropriate third party hardware into the design, incorporating the appropriate attachments into the framework, or adjusting some other portion of the framework 200.

For example, Figure 5A depicts a portion of a framework 200 representing a drawer 500. Specifically, the depicted drawer 500 comprises a backside 522, a left side 532, a right side 530, a front side 520, and a bottom 510. In at least one implementation, the manufacturing preparation module 130 can identify that a front

side 520 comprises a third party hardware interface area 540 (i.e., a location where a handle can be attached). In response, and to identify the third party hardware interface area 540, the manufacturing preparation module 130 can query the configuration list module 160 to identify specifications for a handle that is used by a millwork facility of interest.

In at least one implementation, the configuration list module 160 can access a plurality of different configuration files from different millwork facilities stored on a storage device 150. A designer or user can indicate to the millwork software 100 the particular millwork facility that the user intends to use. The configuration list module 160 can then access the appropriate configuration list 152 (see Figure 1) that is associated with the identified millwork facility. The various configuration lists 152 can be stored within the storage device 150.

Once the configuration list module 160 provides the manufacturing preparation module 130 with the appropriate information, the manufacturing preparation module 130 can identify that the desired millwork facility utilizes handles that are anchored by two screws a certain distance apart. Based upon this information the manufacturing preparation module 130 can place two holes that are the appropriate distance apart within the third party hardware interface area 540 of the front side 520 as shown in Figure 5B. In at least one implementation, the manufacturing preparation module 130 automatically incorporates the changes into a CNC file, or some other format that the desired millwork facility can interpret.

In at least one implementation, instead of selecting a particular millwork facility, the millwork software 100 can provide the user with a selection of all of the available handles. Once a user selects a desired handle, the millwork software 100 can provide the user with a list of the millwork facilities that provide the selected handle. Similarly, in at least one implementation, the user can select a generic handle that can be replaced by an available handle at whatever millwork facility ends up performing the work. Accordingly, the millwork software 100 provides a user with numerous methods for incorporating third party hardware into a particular design.

In addition to incorporating millwork facility specific components into an architectural design, in at least one implementation the manufacturing preparation module can incorporate millwork facility specific joint details into an architectural

design. For example, the configuration list module 160 may identify that a particular millwork facility uses a groove joint to attach the bottom 510 of a drawer 500 to the right side 530 of the drawer 500. Upon receiving this information from the configuration list module 160, the manufacturing preparation module 130 can
5 automatically incorporate the groove joint 532 into the right side 530 of the drawer 500.

Figure 5D depicts the drawer of Figure 5C after the manufacturing preparation module 130 has applied the specified connection type 532 to the drawer. For example, the connection 532 can comprise the bottom side 510 being inset into the
10 right side 530. Upon determining and applying the proper connection type, the manufacturing preparation module 130 can create a file that will direct a millwork facility to cut a groove into the side wall of right side 530, wherein the groove is cut at such a depth and location that the joint functions as designed in Figure 5D.

In at least one implementation, the configuration list 152 provided by the
15 configuration list module 160 can include, among other things, the particular specifications and types of hardware that a particular millwork facility uses, the type of materials and thicknesses of materials that a millwork facility uses, and the types of joint details that the millwork facility uses. Additionally, the configuration list 152 can include information associating at least a portion of the items within the list with
20 particular elements of architectural designs. For example, the configuration list 152 can contain an entry that associates a particular handle with any interface 540 on a drawer front panel 520. Similarly, the configuration list 152 can comprise an entry that associates a groove joint with any interface where a right side drawer panel 530 meets a bottom drawer panel 510.

25 Additionally, in at least one implementation, a configuration list 152 can provide multiple options for, among other things, hardware components, materials type and thickness, and joints. For example, a particular millwork facility may have a plurality of different drawer handles available for use. The configuration list 152 can contain specifications and information about each available handle. The
30 manufacturing preparation module 130 can then provide a user with various options to determine which components and specifications should be incorporated into a corresponding CNC file.

Figure 5E depicts a storage unit 600 that has been designed with an interface 552 for a sink 550. The depicted storage unit 600 comprises a cupboard 602 and three drawers 604, 606, 608. In at least one implementation, a designer can design the storage unit 600 without knowing the specifications for the sink 550 that will
5 eventually be included with the storage unit 600. Additionally, the designer can design the storage unit 600 with the intent that the storage unit 600 be manufactured by a plurality of different millwork facilities, with each millwork facility using a unique sink 550.

The configuration list module 160 can provide the manufacturing preparation
10 module 130 with the appropriate specifications for the depicted sink 550. As depicted in Figure 5E, for example, the sink 550 is too large for the specified interface 552. In at least one implementation, the manufacturing preparation module 130 can resolve this discrepancy by automatically adjusting the size and configuration of the storage unit 600. For example, Figure 5F depicts an adjusted storage unit 600 that now
15 includes the sink 550.

In the illustrated case, the manufacturing preparation module 130 lengthened the cupboard 602 of the storage unit 600 such that it now includes two cupboard doors 602a and 602b. The manufacturing preparation module 130 also narrowed the drawers 604, 606, 608 to compensate for the enlarged cupboard 602. Due to the
20 changes that were automatically made to the storage unit 600 by the manufacturing preparation module 130, the sink 550 now fits. In addition to reconfiguring the storage unit 600, the manufacturing preparation module 130 can also enlarge the interface 552 such that the sink 550 fits. The manufacturing module can also make the changes to a CNC file, or equivalent, such that the specified millwork facility can
25 automatically manufacture the resolved storage unit 600 and sink 550.

In at least one implementation, when resolving inconsistencies within a particular architectural design, the manufacturing preparation module 130 can rely upon a series of predefined constraints. For example, the manufacturing preparation module 130 can be directed to leave unchanged the external boundaries of a particular
30 architectural element. For instance, the storage unit 600 and sink 550 from Figure 5F can have the same external specifications as the original storage unit 600 from Figure 5E. One will understand that, in some cases, adjusting the external boundaries of the

storage unit 600 can prevent the storage unit 600 from fitting in the location for which it was designed.

Additionally, in at least one implementation, the manufacturing preparation module 130 can be directed to adjust an architectural design to incorporate the hardware components specified by the configuration list module 160. For example, one will understand that, in general, a sink 550 cannot be placed directly over a drawer 604 because the sink will extend too deeply into the storage unit 600. Accordingly, in at least one implementation, the manufacturing preparation module 130 can identify that the cupboard 602 (and not the drawers 604, 606, 608) should be expanded to allow the sink 550 to fit.

In determining how to resolve inconsistency within an architectural design, in at least one implementation, the manufacturing preparation module 130 can receive direction from at least one independently executable software object associated with a space 212, 214 within the framework. For example, as described above, a particular space 212 may identify itself as a cupboard 602, while another space 214 may identify itself as drawers 603, 606, 608. Additionally, the independently executable software object associated with the drawers can indicate that an object, such as a sink 550, should not be placed in the same space 214 as the drawers. In contrast, the independently executable software object associated with the cupboard 602 can indicate that an object, such as a sink (e.g., 550), can be placed in the same space 212 as the cupboards.

Once a user has finished designing an architectural element, the manufacturing preparation module 130 can generate one or more CNC codes (or equivalent) that corresponds with or otherwise describes the architectural element. In an alternate implementation, the manufacturing preparation module 130 can generate parameters that are exported to a post processor that generates an appropriate CNC code. Once the appropriate CNC code is generated a millwork facility can use the code to create the designed architectural element.

Figure 6 depicts a finished rendering of the architectural element 200. In particular, Figure 6 shows that the storage unit 600 comprises a cupboard 602 that correlates with the space 212 that was create by the placement of cube splitter 210. In addition, Figure 6 shows that the cupboard 602 comprises a door 610, which, for

example, can be associated with facet 220. In addition, the storage unit 600 comprises three drawers 604, 606, 608 that were created by the combined placement of the cube splitter 210 and the facet splitters 230 and 232. As depicted, facet 234 can be associated with drawer 604, facet 236 with drawer 606, and facet 238 with drawer
5 608. In at least one implementation, additional designing that was not depicted directly by this application may have also been added to the framework 200. For example, the drawer 500 of Figures 5A-5D may have also been designed and added to the framework 200 of Figure 6.

Additionally, in at least one implementation, the system can automatically add
10 gaps to the various components of a framework 200 such that features like drawers 234, 236, 238 and doors 610 are easy to open and close and are not overly snug. In some cases, the gaps may comprise slight millimeter spaces that are incorporated around the edges of a particular facet 212, 234, 236, 238. One will understand that if certain components of architectural elements are not designed and built with a gap,
15 the component may not function (or may function poorly), and such gaps can be easy to overlook during a conventional design face.

In addition to the foregoing, in at least one implementation, the storage device 150 (and/or a configuration list 152) provided by the configuration file module 160 can contain visual information relating to various third party hardware that specific
20 millwork facilities use. As such, in at least one implementation, the user interface module 120 can render a depiction of the architectural element, in this case the storage unit 600, displaying the unit as it will appear in its final form, including the correct connection types and third party hardware.

In at least one implementation, once a user has designed an architectural
25 element, the user can store the design within the storage device 150 for later access. For example, in at least one implementation, this allows the user to incorporate the architectural element into a new design. In particular, the designed framework can be recursively linked to an independently executable software object within another framework.

30 Figures 7A and 7B depict implementations for incorporating one framework (e.g., 200, Figure 2C) into another. For example, Figure 7A depicts a master framework 700 that includes a counter 710, three upper spaces 702, 704, 706, and two

larger lower spaces 730, 732. In at least one implementation, the framework module 140 can associate a distinct framework 200 with a space 730, 732. Specifically, the space module 144 can use the recursion module 148 to associate a distinct framework with the independently executable software object that is associated with the space
5 730, 732. In other words, in at least one implementation, the system associates each space 730, 732 within a framework 700 with an independently-executable software object, which can recursively reference a copy of another distinct framework 200. As previously mentioned, an independently executable software object comprises a set of computer-executable instructions used in object-oriented program code, and which
10 relate to a particular physical component or feature.

Returning to Figures 7A and 7B, a user can associate the two distinct frameworks 200 each representing the storage unit 600 from Figure 6 with spaces 730 and 732 respectively. Figure 7B depicts the resulting master framework 700 that includes spaces 702, 704, and 706, which can be designed into cupboards, and spaces
15 730 and 732, which both now contain frameworks 200 that are associated with storage units 600.

In at least one implementation, associating distinct frameworks 200 with spaces 730, 732 within a master framework provides a user with tremendous power and flexibility in creating a design. For example, each framework 200 can
20 independently access a framework module 140 and all other associated modules 142, 144, 146, 148. This can allow a framework 200 to dynamically and automatically adjust to any changes that are made to a master framework 700.

Additionally, in at least one implementation, a storage device 150 can comprise a framework library of pre-designed architectural elements. Each of these
25 stored frameworks can be associated with one or more independently executable software objects that can be recursively linked to other frameworks. For example, a designer can design an office space by accessing a group of stored frameworks that represent shelving units, desks, filing cabinets, cupboards, drawers, etc.

Once a designer identifies particular stored frameworks that the designer
30 wants to use, the designer can simply insert the chosen framework into a space within a master framework. Using pre-designed frameworks a designer can create a master framework that represents an entire office. Additionally, because the entire office was

designed using spatial frameworks associated with independently executable software objects, the entire office design can change automatically to account for different sizes, materials, features, etc., and such changes will correctly propagate throughout the design.

5 For example, Figure 8A depicts the cabinet system 800 designed in Figures 7A and 7B. The cabinet system 800 comprises a length 810, and a receiving wall 830 comprises a length 820. As depicted, length 810 is significantly longer than length 820 of the receiving wall. In at least one implementation, a user can specify that the length of the cabinet system 800 should be length 820. The framework module 140,
10 in turn, can automatically adjust the length of the cabinet system 800 to be length 820, while at the same time automatically and correctly adjusting all of the features of the cabinet system.

 Figure 8B depicts an implementation of an adjusted cabinet system 800. As shown, the system (e.g., via framework module 140) automatically removed a
15 cupboard and one of the storage units. In at least one implementation, the removal of the cupboard and storage unit may be a result of using absolute measurements when tracking the location and behavior of the cube splitters 210 and facet splitters 230, 232, such as described with respect to Figure 3. In particular, the framework module 140 may have identified that the storage unit length was reduced so much that there
20 was no longer room to place the cube splitters 210 and facet splitters 230, 232 as was initially specified. Accordingly, the system (e.g., via framework module 140) can automatically determine that, because the original length specifications cannot be met, a cupboard and a storage unit (represented by framework 200 in space 732) should be removed.

25 As an alternate example, Figures 9A and 9B depict an architectural element being expanded to fit a particular specification. For example, the cabinet system 800 comprises a length 810, and a receiving wall 930 comprises a length 910. As depicted, length 810 is significantly smaller than length 910 of the receiving wall 930. As stated above, in at least one implementation, a user can specify that the length of
30 the cabinet system 800 should be 910. The framework module 140, in turn, can automatically adjust the length of the cabinet system 800 to be length 910, while at

the same time automatically and correctly adjusting all of the features of the cabinet system.

Figure 9B depicts an implementation of an adjusted cabinet system 800. In this implementation, the system (e.g., via framework module 140) automatically
5 expanded the length of the storage units 600 and added double doors 940 to each of the three cabinets. In at least one implementation, the expansion of the storage unit may be a result of using proportional measurements when tracking the location and behavior of the cube splitters 210 and facet splitters 230, 232 as described with respect to Figure 3. In particular, the framework module 140 may have identified that
10 the cube splitter between spaces 730 and 732 was specified as being placed at half the length of the bottom most boundary. As such, the framework module 140 simply expanded the storage units to fill the larger length.

With respect to the cupboards, in at least one implementation, the framework module 140 and/or the manufacturing preparation module 130 can automatically
15 identify some finishing features, such as, for example, whether a cupboard is narrow enough to only require a single door, or so wide as to require double doors 940. In the depicted example, the system (e.g., via framework module 140 and/or the manufacturing preparation module 130) determined that double doors were appropriate due to the increased length of the cabinet system 800.

20 Accordingly, Figures 1-9 and the corresponding text illustrate or otherwise describe one or more components, modules, and/or mechanisms for automatically resolving boundaries. One will appreciate that implementations of the present invention can also be described in terms of methods comprising one or more acts for accomplishing a particular result. For example, Figures 10 and 11 and the
25 corresponding text illustrate flowcharts of a sequence of acts in a method for resolving boundaries within an architectural design. The acts of Figures 10 and 11 are described below with reference to the components and modules illustrated in Figures 1-9.

For example, Figure 10 illustrates that a method for resolving boundaries
30 within architectural design can comprise an act 1000 of receiving a digital architectural design. Act 1000 includes receiving a digital architectural design comprising a first furniture sub-component, a second furniture sub-component, and a

third furniture sub-component. For example, in Figure 4A and the accompanying description, the millwork software 100 receives an architectural design that comprises a first furniture sub-component 400, a second furniture sub-component 402, and a third furniture sub-component 404. The design depicted in Figure 4A may be the
5 initial design of a desk, a chest, shelf, or some other similar furniture piece. In any case, the millwork software 100 can receive an architectural design that comprises multiple sub-components.

Figure 10 also shows that the method can comprise an act 1010 of identifying joints within the design. Act 1010 includes identifying, with a computer processor, a
10 first joint between the first furniture sub-component and the second furniture sub-component, a second joint between the second furniture sub-component and the third furniture sub-component, a third joint between the third furniture sub-component and the first furniture sub-component. For example, Figure 1 depicts a boundary module 146 that is configured to identify boundaries within an architectural design. For
15 instance, Figure 4A illustrates boundaries 410, 420, and 430, which can each be identified by the software through boundary module 146.

Additionally, Figure 10 also shows that the method can comprise an act 1020 of identifying an anomaly. Act 1020 includes identifying an anomaly at the intersection of the first joint, the second joint, and the third joint, wherein the anomaly
20 is created when the first joint, the second joint, and the third joint fail to create a proper corner. For example, Figure 4B shows an anomaly 440 that occurs at the intersection of a particular arrangement of surfaces 400, 402, and 404. The boundary module 146 can automatically identify this anomaly 440 by identifying a gap at the illustrated joint.

Further, Figure 10 shows that the method can comprise an act 1030 of automatically resolving the anomaly. Act 1030 can include automatically resolving the anomaly by changing the shape or orientation of at least one of the first joint, the second joint, or the third joint within the digital architectural design. The Act can also include sending rendering instructions to a display device for visually displaying the
25 corrected anomaly. For example, Figure 4B and the accompanying text describe and depict an architectural design with an anomaly 440. The boundary module 146 can engage in various methods and actions to automatically resolve the anomaly. For
30

instance, the software 100 can adjust the surfaces 400, 402, and 404 of Figures 4A 4B (whether adjusting position, sizing, or the like, as appropriate), such that the front panel 402 overlaps the side panel 404 to correct the anomaly. The software 100 can then display the corrected joint.

5 Additionally, Figure 11 illustrates an additional or alternative method for resolving boundaries within architectural design can comprise an act 1100 of analyzing a digital architectural design. Act 1100 includes analyzing, with a computer processor, digital architectural design input received from a user. The digital architectural design input can comprise a first furniture sub-component, a
10 second furniture sub-component, and a third furniture sub-component. The first furniture sub-component, the second furniture sub-component, and the third furniture sub-component can each be associated with one or more independently executable software objects. For example, upon receipt of appropriate input, the software 100 can automatically identify that an architectural design comprises a first furniture sub-
15 component 400, a second furniture sub-component 402, and a third furniture sub-component 404. As described in the accompanying description, one or more of the sub-components 400, 402, 404 can be associated with an independently executable software object.

Figure 11 also shows that the method can comprise an act 1110 of identifying
20 joints within the design. Act 1110 includes identifying a first joint between the first furniture sub-component and the second furniture sub-component, a second joint between the second furniture sub-component and the third furniture sub-component, a third joint between the third furniture sub-component and the first furniture sub-component. For example, Figure 1 depicts a boundary module 146 that is configured
25 to identify boundaries within an architectural design. For instance, in Figure 4A and the accompanying description, the software 100 (e.g., via boundary module 146) can identify boundaries 410, 420, and 430.

Additionally, Figure 11 shows that the method can comprise an act 1120 of identifying an anomaly. Act 1120 includes identifying an anomaly at the intersection
30 of the first joint, the second joint, and the third joint, wherein the anomaly is created when the first joint, the second joint, and the third joint fail to create a proper corner. For example, as discussed with respect to Figures 4A and 4B, the software 100 (e.g.,

the boundary module 146) can identify anomaly 440, such as by identifying a gap in a corner where various surfaces or sub-components meet improperly.

Figure 11 also shows that the method can comprise an act 1130 of determining one or more compatible joint specifications. Act 1130 can include determining one or more joint specifications that are compatible with the architectural design from information retrieved from the one or more independently executable software objects. For example, as described with respect to Figure 4B, the software 100 (e.g., via the boundary module 146) can access an independently executable software object (not shown) to receive information such as material type, dimensions, strength requirements, and other related characteristics that determine compatible joint specifications. For instance, upon determining that the various furniture sub-components are constructed of marble, the boundary module 146 may determine that overlap joints should be used to more easily display the marble.

Further, Figure 11 shows that the method can comprise an act 1140 of automatically resolving the anomaly. Act 1140 can include creating an updated digital architectural design with an automatic resolution of the anomaly by changing the type of at least one of the first joint, the second joint, or the third joint to comprise a revised joint that conforms with the one or more joint specifications. For example, Figures 4A and 4B and the accompanying description describe and depict an architectural design with an anomaly 440. In at least one implementation, the software 100 (e.g., via boundary module 146) can change boundary 422, such that the front panel 402 overlaps the side panel 404, thus making all of the seams meet up perfectly, and which would correct the anomaly.

Further still, Figure 11 shows that the method can comprise an act 1150 of creating an updated millwork file. Act 1150 can include storing the updated digital architectural design in a format specific to a millwork facility. For example, Figure 1 and the accompanying description describes and depicts a manufacturing preparation module 130 disposed within the millwork software 100. The manufacturing preparation module 130 can architectural design, including the revised joint, and create a file that is formatted for manufacturing at a millwork facility.

Accordingly, Figures 1-11 and the corresponding text illustrate or otherwise describe a number of components, schematics, and mechanisms for incorporating

automatically resolving boundaries within an architectural design. One will appreciate that these components and modules in accordance with implementations of the invention can allow a designer to develop an architectural element, such as a desk or even an entire kitchen, without knowing the specific joints that are used by a particular millwork facility. For example, a user can design a kitchen, including cabinets, drawers, counter tops, sink locations, etc., without knowing the specific joints that will be used to construct the cabinet and counter. Additionally, these and other implementations of the invention can be used to identify an anomaly within an architectural design before the design is manufactured at a millwork. Various modules, such as boundary module 146, can then automatically correct a corner anomaly without any user interaction, or with only minimal user interaction.

In addition to the foregoing, one will appreciate that embodiments of the present invention may comprise a special purpose or general-purpose computer including various computer hardware components, as discussed in greater detail below. Embodiments within the scope of the present invention also include computer-readable media for carrying or having computer-executable instructions or data structures stored thereon. Such computer-readable media can be any available media that can be accessed by a general purpose or special purpose computer.

By way of example, and not limitation, such computer-readable media can comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to carry or store desired program code means in the form of computer-executable instructions or data structures and which can be accessed by a general purpose or special purpose computer. When information is transferred or provided over a network or another communications connection (either hardwired, wireless, or a combination of hardwired or wireless) to a computer, the computer properly views the connection as a computer-readable medium. Thus, any such connection is properly termed a computer-readable medium. Combinations of the above should also be included within the scope of computer-readable media.

Computer-executable instructions comprise, for example, instructions and data which cause a general purpose computer, special purpose computer, or special purpose processing device to perform a certain function or group of functions.

Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described above. Rather, the specific features and acts described above are
5 disclosed as example forms of implementing the claims.

The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the
10 foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

CLAIMS

I claim:

1. In a computerized environment comprising a computer system that receives user
5 design input, a computerized method for automatically resolving joint anomalies within
digital joints of an architectural design, wherein such digital joints are representative of
physical joints manufacturable in a millwork facility, the method comprising:

receiving a digital architectural design comprising a first furniture sub-
component, a second furniture sub-component, a third furniture sub-component;

10 identifying, with a computer processor, a first joint between the first furniture sub-
component and the second furniture sub-component, a second joint between the second
furniture sub-component and the third furniture sub-component, a third joint between the
third furniture sub-component and the first furniture sub-component, wherein the
received digital architectural design includes a user-provided particular type of joint
15 corresponding to each of the first joint, the second joint, and the third joint;

identifying an anomaly at the intersection of the first joint, the second joint, and
the third joint, wherein the anomaly is created when the first joint, the second joint, and
the third joint fail to create a proper corner; and

20 automatically resolving the anomaly by changing the shape or orientation of at
least one of the first joint, the second joint, or the third joint within the digital
architectural design, and sending rendering instructions to a display device for visually
displaying the corrected anomaly.

2. The method as recited in claim 1, wherein each of the first furniture sub-
25 component, the second furniture sub-component, and the third furniture sub-component is
associated with a respective independently executable software object.

3. The method as recited in claim 2 further comprising:

accessing the independently executable software object that is associated with the second furniture sub-component;

determining one or more types of joints that are compatible with the second furniture sub-component; and

changing the type of the second joint to one of the one or more types of compatible joints.

4. The method as recited in claim 2 further comprising:

accessing the independently executable software object that is associated with the third furniture sub-component;

determining one or more types of joints that are compatible with the third furniture sub-component; and

changing the type of the third joint to one of the one or more types of compatible joints.

5. The method as recited in claim 1 further comprising:

determining that the first joint is preferred; and

automatically resolving the anomaly by changing the type of at least one of the second joint or the third joint.

6. The method as recited in claim 5 wherein determining that the first joint is preferred comprises receiving an indication from a user that the first joint is preferred.

7. The method as recited in claim 5 wherein determining that the first joint is preferred further comprises determining that the first joint was created before the second joint and the third joint.

8. The method as recited in claim 5 wherein determining that the first joint is preferred further comprises determining that the first joint was created after the second joint and the third joint.

5 9. The method as recited in claim 5 wherein determining that the first joint is preferred further comprises:

identifying one or more additional joints within the architectural design;

determining a joint type associated with each of the one or more additional joints;

and

10 determining that the first joint is the same type of joint as the one or more additional joints.

10. The method as recited in claim 1 further comprising:

generating a computer numerical code file that corresponds with the architectural
15 design, including the changed first, second, or third joint.

11. In a computer-based system comprising one or more processors for receiving and processing user design input, a computerized method for automatically resolving joint anomalies within digital joints of an architectural design, wherein such digital joints are representative of physical joints manufacturable in a millwork facility, the method comprising:

5 analyzing, with a computer processor, digital architectural design input received from a user, the digital architectural design input comprising a first furniture sub-component, a second furniture sub-component, and a third furniture sub-component, wherein the first furniture sub-component, the second furniture sub-component, and the third furniture sub-component are each associated with one or more independently
10 executable software objects;

identifying a first joint between the first furniture sub-component and the second furniture sub-component, a second joint between the second furniture sub-component and the third furniture sub-component, a third joint between the third furniture sub-component and the first furniture sub-component, wherein the received digital
15 architectural design includes a user-provided particular type of joint corresponding to each of the first joint, the second joint, and the third joint;

identifying an anomaly at the intersection of the first joint, the second joint, and the third joint, wherein the anomaly is created when the first joint, the second joint, and the third joint fail to create a proper corner;

20 determining one or more joint specifications that are compatible with the architectural design from information retrieved from the one or more independently executable software objects;

creating an updated digital architectural design with an automatic resolution of the anomaly by changing the type of at least one of the first joint, the second joint, or the
25 third joint to comprise a revised joint that conforms with the one or more joint specifications; and

storing the updated digital architectural design in a format acceptable to a millwork facility.

30 12. The system as recited in claim 11 further comprising:

determining that the first joint is preferred; and

automatically resolving the anomaly by changing the type of at least one of the second joint or the third joint.

13. The system as recited in claim 12 wherein determining that the first joint is preferred comprises receiving an indication from a user that the first joint is preferred.

14. The system as recited in claim 12 wherein determining that the first joint is preferred further comprises determining that the first joint was created before the second joint and the third joint.

10

15. The system as recited in claim 12 wherein determining that the first joint is preferred further comprises determining that the first joint was created after the second joint and the third joint.

16. The system as recited in claim 12 wherein determining that the first joint is preferred further comprises:

identifying one or more additional joints within the architectural design;
determining a joint type associated with each of the one or more additional joints;
and

determining that the first joint is the same type of joint as the one or more additional joints.

17. The system as recited in claim 11 further comprising:
identifying one or more additional joints within the architectural design;
determining the type of joint associated with each of the one or more additional
joints; and
5 automatically resolving the anomaly by changing the type of at least one of the
first joint, the second joint, or the third joint to comprise a revised joint that matches one
of the determined type of joints.

18. The system as recited in claim 17 wherein the one or more additional joints within
10 the architectural design are joints adjacent to the first joint, the second joint, or the third joint.

19. The system as recited in claim 17 wherein the one or more additional joints within
the architectural design are joints framing the same furniture sub-component as the first joint, the
second joint, or the third joint.

15

20. In a computerized environment comprising one or more computer systems having one or more processors, one or more physical storage media comprising computer executable instructions that, when executed by the one or more processors cause the one or more computer systems to automatically resolve joint anomalies within digital joints of an architectural design, wherein such digital joints are representative of physical joints manufacturable in a millwork facility, the one or more processors performing the following:

receiving a digital architectural design comprising a first furniture sub-component, a second furniture sub-component, and a third furniture sub-component;

identifying a first joint between the first furniture sub-component and the second furniture sub-component, a second joint between the second furniture sub-component and the third furniture sub-component, a third joint between the third furniture sub-component and the first furniture sub-component, wherein the received digital architectural design includes a user-provided particular type of joint corresponding to each of the first joint, the second joint, and the third joint;

identifying an anomaly at the intersection of the first joint, the second joint, and the third joint, wherein the anomaly is created when the first joint, the second joint, and the third joint fail to create a proper corner; and

automatically resolving the anomaly by changing the type of at least one of the first joint, the second joint, or the third joint within the digital architectural design, and sending rendering instructions to a display device for visually displaying the corrected anomaly.