Superficial – the concepts

Superficial is a fresh approach to the design and coding of interactive applications, in particular those with graphical user interfaces (GUIs).

Superficial

- provides a conceptual framework for describing a user interface and how it exposes application content;
- enables a strong separation of concerns that simplifies development, debugging and maintenance of GUI applications;
- largely eliminates the complexities of data binding and event listening that arise with conventional approaches to GUI design;
- has been implemented in the Java programming language and could probably be implemented in any object-oriented language.

This paper gives a detailed account of the Superficial approach, aimed primarily at readers considering development of production-quality code based on Superficial.

The concepts defined by Superficial are introduced under the following headings:

- The **metaphor** of *surface* provides a basis for describing any interactive application.
- The **analysis** applies the surface metaphor to GUI applications in particular, abstracting the widgets of a GUI surface into *facets* connected to *targets* within the application.
- The **architecture** turns the analysis to practical use by defining a mediating tree of *targeters* that ensures facets are always connected to appropriate targets.
- The **mechanism** uses the architecture to define a *build* sequence that assembles a surface attached to a targeter tree; and a *retargeting* sequence that maintains consistency between surface and application.

Practical **implementation** of these concepts in a language such as Java requires further design decisions and completing components; **production use** of Superficial could be based on further development of the Facets demonstration implementation.

At the end of this paper there is a **glossary** of Superficial terminology

Interaction implies surface – the metaphor

The cornerstone of the Superficial approach is the metaphor of surface.

A software application can be defined as comprising content from a problem domain together with logic about how that content can change. To interact with an application the user must be able to access its content, request changes subject to the constraints of the logic, and access the results of such changes. Since the user is outside the application and the content is on the inside, the means by which the user interacts with the application can helpfully be thought of as its *surface*.

This view of interaction resembles in some respects the Model-View-Controller paradigm (MVC). A surface is indeed most clearly defined as an application element that exposes content to *view* and *control* by the user, where view enables the user to determine the state of the content, and control enables them to modify that state.

However the surface metaphor differs from MVC in important ways. Subsuming view and control as subordinate concepts, it is readily applied to GUI applications where the majority of widgets provide both functions. It also defines a distinct problem domain for interactive applications themselves: the building and management of their surfaces.

Superficial takes a broad view of the model, treating it as coextensive with the complete application. This provides for the common case where some elements of a GUI surface expose other elements to user view and control: for instance, a pull-down menu that changes window focus or layout.

Applying the metaphor – the analysis

For the surface metaphor to apply usefully to GUI applications, a way must be found to analyse their surfaces.

Such an analysis must address two major issues. Firstly, a GUI surface is constructed from what can be a large number of widgets. Many of these widgets define purely decorative features such as labels or borders, which provide neither view nor control of content but must still be specified as part of the surface.

The second difficulty is that the logical structure of an application rarely maps directly onto the widget structure of its GUI surface. Several widgets may expose the same application element to view and control; or the element exposed may need to be replaced by another, for instance following a change in content selection.

How Superficial deals with the second difficulty is described later; the first is handled by analysing a GUI surface into elements that abstract away its concrete widgets, while still defining its visual layout.

Superficial calls such abstract surface elements *facets*, expressing their character as discrete elements of the complete surface. Each facet manages one or more widgets that form a visual and logical group; a GUI surface can thus be regarded as comprised of an outer layer of concrete widgets, managed by an inner layer of

abstract facets. The widgets of the outer layer are visible to the user but invisible to the application; their managing facets in the inner layer are visible to the application but not literally speaking to the user.

However since the visual and logical layout inferred by the user from the widget layer should resemble closely that defined by the facet layer, the facet layer largely defines how the surface will be perceived by the user.

The facets of a surface can be divided into two categories:

- *simple facets* which expose primitive data values, actions and fixed content structures in the application;
- *viewer facets* which expose looser (and usually more complex) content structures.

Simple facets manage the widgets of menus, tool bars, dialogs etc; viewer facets are used for the content area of most applications as well as for dialogs such as file finders and property editors.

Simple facets

Simple facets, of which the simplest useful example is probably the checkbox, share a number of features. A simple facet

- exposes to user view and control a *target* in the application: usually a single data value roughly equivalent to a language primitive, sometimes an action that the application can perform.
- is often grouped with other related facets: an XY coordinate pair for instance, or file manipulation actions.
- is identified visually by some combination of text labels and graphics such as icons or borders.
- is laid out within containers such as panels, menus, tool bars etc that may not themselves have targets.
- though it exposes a simple value in data terms, may nonetheless have complex variants – for example the tri-state checkbox.

Perhaps surprisingly, a simple facet can always be defined in terms of a very limited set of target types, for which Superficial coins a few more terms.

- Checkboxes, toggle buttons and their menu equivalents expose *toggling* targets representing Boolean values.
- Sliders, some formatted text fields and nudging button pairs expose *numeric* targets.
- Plain and formatted text fields expose *textual* targets, as do non-decorative labels such as status lines.

- Single-selection lists and dropdowns expose *indexing* targets which are indexes into lists; as do iterating button pairs, some groups of menu items and (less obviously) radio-button groups.
- Buttons and menu items not directly connected to data values expose *trigger* targets that initiate an application process.

More elaborate simple facets can be treated as exposing groupings of these types. A spinner combines a formatted text field with iterating buttons to expose a numeric grouped with an indexing. A multi-selection list exposes a group of togglings, one for each item in the list.

Describing facets in terms of their targets can shed light on their functional nature. For example, an editable combo or other combination of list and text field can be analysed as exposing a textual grouped with an indexing, behaving in different ways depending on the functional relationship between the two. The textual may be editable

- intermittently, to rename items in the indexing;
- at all times, but only as an aid to navigating the indexing;
- freely, with the indexing providing a suggestion list.

Simple facets can be grouped further as required to expose any target that has a fixed structure. This means that a surface that exposes content comprised of one or more fixed data structures can consist purely of groups of simple facets; such surfaces are frequently used for an application dialogs.

Viewer facets

All content is composed ultimately of fixed structures of primitive data values. However the content of many applications has a looser overall structure which must be exposed by means of viewer facets.

While based on the same concepts of facet and target, a viewer facet differs substantially from a simple facet. A viewer facet

- enables the user to view and manipulate *avatars* depicting its loosely structured content target.
- is not grouped with others like a simple facet, though a content area may contain several viewers.
- will often be itself controlled (activated/shown/hidden) by other facets in menus or tool bars.
- usually provides additional direct control of its target from the keyboard, and from facets in a context-sensitive menu and elsewhere in the surface.

There are a number of standard avatar styles such as tree, table and text flow, and even a complex surface may need only some combination of viewers with these avatars. However the set of viewer facets is not more or less closed as for simple

facets, since many content types require viewers with custom avatars; and even the avatars of standard viewers are usually configured to suit their content.

Since viewers using arbitrary avatars may expose arbitrary content, it is impractical to analyse them like simple facets in terms of their target type. Instead Superficial uses three abstracting types to define what the user interacts with in a viewer:

- the *viewer* facet itself, managing a viewer widget specified by its avatar style
- the *viewable* content that the widget is to expose using this avatar style
- the *view* policy that specifies how the widget should expose the viewable

A further abstraction is needed for the containing facets of viewers, such as panes and windows. As the container of either a single viewer or other containing areas, the *area* naturally forms a hierarchy rooted in the top-level surface window.

Further complications arise when defining the targets associated with viewer facets. Firstly, the viewer itself really has two targets: the appearance and behaviour of its avatars depend as much on its view policy as on its viewable content.

Secondly, the viewable, view and the viewer itself may supply targets in their own right to other facets. Elements of the current selection within the viewable will typically be exposed by other facets in the surface. A view may have variable properties, such as zoom or display of gridlines, that are targeted by other simple facets. Direct content actions defined for the viewer must be exposed by trigger facets.

The containing areas for viewers may also be targets for other facets or for the internal logic of the surface. In particular an area facet must represent to the application the pane or window that it manages, so that this can be given focus either programmatically or by user interaction with other facets such as menus; and so that focus change in the widget surface layer can be relayed by the facet layer to the application.

The complete surface

The analysis of a GUI surface into simple facets and their groupings, viewer facets and their containing areas allows Superficial to define clearly the required characteristics of a complete surface.

The overriding requirement for any surface is consistency of view and control. To maintain transparency of user interaction, any change in either application content or the surface itself must be reflected immediately in all facets.

Maintaining view and control consistency in a GUI surface is complicated by the need to provide for two further surface characteristics:

- multiple facets exposing a single target
- content selection

For both simple and viewer facets, more than one facet in the surface may expose the same target in the application. A simple target may be exposed by several functionally identical sets of simple facets. One set may appear grouped in a toolbar and another (most probably of different styles) on a sidebar panel; a further ungrouped set may even be mixed with those exposing other targets in a menu tree. Viewable content may be the target of several viewer facets, each displaying different regions and/or different views of the same content.

For view and control consistency to be maintained, all simple facets exposing a given target must provide the same view and control of that target; and the case of viewers is once again more complex. Not only must they provide consistent view and control of what may be different portions of the same content, but user interaction with one viewer may trigger a change in the view policy of others.

Surfaces must also support selection within the content they expose. The selection may be defined by simple facets such as navigation buttons providing traversal of records, or by the user activating a viewer and selecting within the avatars depicting its content.

Multiple selection may be required. Though not useful in text processing, it can be necessary for tree or table viewers and is more or less essential for manipulating graphics. This makes complex demands on the behaviour of all facets. If the current selection is multiple, even simple facets must be able to respond either by becoming inactive or by providing appropriate view and control (such the tri-state checkbox already mentioned).

Whether single or multiple, each selection change will require much of the surface as a whole to be updated. Both simple and viewer facets must respond to the new selection; and may also themselves need to be enabled or disabled, shown or hidden.

The Superficial approach to updating the surface is that any change in the application which might affect view and control consistency should trigger a *retargeting*. During retargeting all facets in the surface are updated either with the state of their existing targets, or in the case of selection change with that of their new targets. Because retargeting provides a general means of updating the surface, it can also be used to bind the surface initially to the application.

It is through retargeting that Superficial handles both the major issues of building and managing a GUI surface: data binding and event listening. Data binding is simplified because retargeting ensures that each facet binds its widgets to the latest state of the content represented by its target. Rather than register piecemeal the interest of facets in specific events, Superficial assumes that any facet may be interested in any event, leaving the facet to decide upon retargeting whether or not to update its widgets..

Such a broad brush approach may appear inefficient, but retargeting and checking the need for widget update is a very lightweight activity compared to repainting updated widgets.

It is notable how the surface metaphor describes equally well a complete surface and its composing facets. Whether simple or viewer, each facet behaves in miniature like the surface as a whole, providing both view and control of the target that it exposes. Of the simple facets only a textual status line provides view of its target without control, and no facet provides control without at least some sort of view. While a simple trigger button or menu item exposes no content, it nonetheless provides view of the fact that its targeted action exists; and it will seriously annoy the user if it fails to provide view, by disabling itself, of the action's non-availability.

From analysis to application – the architecture

To summarise how Superficial analyses a GUI surface:

1. The surface exposes application content to user view and control.
2. An outer surface layer of GUI widgets is visible to the user, managed by an inner layer of facets visible to the application.
3. Simple facets expose targets in the application that are data primitives or actions; groupings of such facets expose fixed content structures; viewer facets expose loose content structures.
4. Some facets (especially viewers) may be controlled by others.
5. Multiple facets in the surface may expose a single target in the application.
6. Some facets may define content selections to be exposed by the surface as a whole, including multiple selections; all facets may need to respond to selection change.
7. Consistency of view and control between facets and targets is maintained by retargeting the surface on the application.

This analysis should in principle apply to any surface that can be constructed using GUI widgets. The only surfaces for which this is proven are those of the Superficial demo applications, but thought experiments suggest that the analysis holds for surfaces as varied as those of Word, Excel, CorelDRAW and the Eclipse platform. Readers may wish to test the analysis on an application they use frequently, or one for which they have coded (or would like to code) a GUI.

The analysis deliberately ignores the question of how the facet layer creates the concrete GUI layer; this requires a *host* that can provide a GUI context within which concrete widgets can be created and laid out.

Targets and trees

Having devised an analysis of GUI surfaces, the next stage is to embody it in an architecture that enables facets to communicate reliably with their targets in the application.

Rather than connect facets directly to content, Superficial defines abstract target types based on those defined for facets, and manages communication between concrete GUI widgets and the application in terms of mediation between facets and targets.

The following target types are defined:

- A base type with general features such as having a human-readable title, knowing whether it is enabled or disabled. It can also contain child target elements: this enables groupings of targets to be constructed to correspond with groupings of facets, and assembled into complete target trees representing regions of the application and its content.
- Variants of the base type for use with simple facets as described earlier. These are facades that allow toggling facets to set flags in the application, numerics to set values, etc, without direct knowledge of the application elements represented by their targets.
- A variant that acts as a *frame* around a discrete application element, allowing direct access to this element such as is required by viewer facets; child target elements may represent elements of the frame content and actions on it.
- Variants of the frame target to represent viewable content, viewers and their containing areas as described earlier.

From these types can be built target trees representing both application content and the windows, panes and viewers of the surface itself; these trees can be assembled into a single tree containing targets for all the facets composing the abstract surface layout.

Such a tree turns out to have a constantly varying structure, because the targets to be exposed by the surface at any given instant map only partially onto the set of potential targets in the application. Different viewer, view and viewable trees must be exposed by the surface as different viewer facets are activated. The selection tree, typically exposed by more facets than any other, must at a minimum be adjusted in some way on each selection change, and is in practice most simply reconstructed from scratch. Even the area target tree may change as windows and panes are opened and closed.

Mediation between surface and targets

As facets have a fixed relationship to their widgets, so do targets (though they may themselves be transient) to the content they represent in the application. For this reason communication between facets and widgets, and between targets and the application, are details of implementation that need not be considered at the architectural level.

However the reverse is the case for the retargeting mediation required between facets and targets. An effective architecture for such mediation is crucial if the surface metaphor is to be applied successfully to GUI applications.

An example of the mediation required between a facet and its target is given once again by the checkbox. The toggling facet managing a concrete checkbox widget exposes to user view and control a toggling target, which itself represents a flag of some sort in the application.

The facet obtains from its target the information necessary to set a suitable label on its widget, to set the widget's toggle state and whether it should be enabled or disabled. If notified by its widget of user input, it can check the new state of the widget and relay that state to its target, which in turn updates the application flag.

It may be that the facet exposes a target forming part of a selection, so that as the selection changes, the facet must be retargeted with a new target. This is simple for the facet, which need only update its checkbox widget with information from its new target. Once retargeted, the facet can respond to input from its widget by setting the new target's state as it did for the previous one.

What is simple for the facet is far from simple for its retargeting mediator, which has to handle both the transience of the complete target tree and the fact that many of its members are exposed by multiple facets in the surface.

Much the commonest cause of change in the target tree is that the viewable frame around the active content updates its selection target tree in response to change either in selection or in the state of the existing selection. In the simplest case the mediator must obtain from the latest selection tree the new toggling target to be exposed by the facet; more complex cases include those where the selection is multiple or of varying type.

While the toggling facet has just one target at a time, it will very likely share this with several other facets, each exposing the target with its own widget (possibly also a checkbox, eqully possibly a toggle button or toggling menu item). The mediator must know of all facets that are thus multiplexed to its target, so as to retarget each with the same new target. Only thus can it ensure that each facet will update its widgets with the correct target information and relay input from those widgets to the correct target.

The targeter tree

The Superficial architecture handles mediation during retargeting by defining a *targeter* type which is in some respects like a facet, in others like a target. A targeter has four principal features:

1. Like a facet, it is *targetable* on a member of a target tree.
2. It maintains a list of facets exposing its target, thus resolving the multiplexing problem described earlier.
3. Like a target, it can contain child elements to match any child elements of its target.
4. Its type is defined by its potential targets: instances are generally obtained from such a target during the initial retargeting.

During retargeting a targeter can retarget any child elements on the child elements of its new target, which means that a tree of targeters can retarget itself and its attached facets on any matching tree of targets.

Since targeters are created from their initial targets, targeter trees can be dynamically generated from target trees. During initial retargeting the root of any target tree can be queried for a suitable targeter, which is then retargeted on the target that created it. The targeter queries its new target for any child elements, queries these in turn for new targeters to become its own child elements, and retargets the new elements on the target elements that created them.

The targeter's new elements repeat this sequence until the complete target tree has been visited and a matching targeter tree constructed. As surface facets are created and attached to targeters, each is retargeted on its targeter's initial target.

In subsequent retargetings each targeter in a tree will generally find that its child elements match those of its new target. Once targeters with attached facets have retargeted these on their current targets, the GUI widgets managed by each facet will expose the latest state of its target and thus of the application element represented by that target.

As with target trees, the targeter tree connecting facets to their current targets may vary in response to changes in the target tree; with the important difference that rather than constantly creating new members, it adjusts its structure by recycling existing members and their attached facets.

By defining which facets expose which targets, a targeter tree adds logical structure to the visual structure of a facet tree and connects this mostly permanent tree to the transient target tree. The key element of a Superficial surface is therefore its targeter tree, primarily composed of one or more *root targeters* that mediate between targets representing content and facets exposing this content to user view and control.

Surfaces and applications

The simplest possible Superficial application has a single surface, with a targeter tree that is the single root targeter tree exposing its content.

However most practical applications need multiple surfaces, and the targeter tree for each surface may contain more than one root targeter. Even if an application exposes only a single content document at a time, it will almost certainly use modal dialogs to expose the properties of the current selection or user preferences. Such dialogs are in effect applets running inside the parent application, with surfaces independent of the parent surface.

Both application and dialog surfaces may need to expose multiple content. A multiple-document application must allow the user to open new tabs or windows on different content. A dialog may have multiple tabs with different content, or even multiple pages which themselves contain multiple tabs.

A surface area tree is often therefore not a single content tree, but a structure of several such trees. In the case of an application it must be possible to add and remove content trees from the surface tree root as the user opens and closes documents; all such trees must share the targeter tree to which are attached the menu and toolbar facets used to expose their content. The surface tree of a dialog is by contrast fixed and consists of independent content trees, but these may be in arbitrary arrangements.

A final complication is that a multi-document application may need to allow existing content to be exposed in new tabs or windows. Each has its own content tree, which must share not only a targeter tree but also the existing content.

Superficial deals with these issues by extending the use of area trees to arrange not just viewers, but complete content trees obtained from *contenters* which wrap the content itself (and usually create it from an external source such as a file or database connection). A contenter can create any number of area trees from its content; once menu and other facets have been attached to the targeter tree created dynamically from the first such tree, they can be shared by all content area trees of that type.

Interaction between elements

Because Superficial integrates an application with its surface purely in terms of facets, targets, and the mediating targeter tree, the interactions of surface elements with each other and with the application are very clearly defined.

- A widget in the concrete GUI interacts with its managing facet only to notify it of input.
- A facet interacts most widely with other elements, needing fairly detailed knowledge of both its widgets and its target. However only viewer facets interact directly with the content framed by their targets; the facades provided by the targets of simple facets shield these facets from direct knowledge of the content they represent.
- A targeter interacts with both its attached facets and its own target, but through very narrow interfaces. This makes it relatively simple for targeter variants to deal with complexities such as multiple selection targets or targets of varying type.
- A target interacts actively only with the application element it represents: it is entirely passive with respect to both the targeter and facets of which it is a target. Even its interaction with the application can be mediated by a separate coupling object as described later in this paper.
- Finally, application elements need have no knowledge whatsoever that they are the subject of interaction. Superficial applications can thus be constructed for content types that make no explicit provision for user interaction.

This strong separation of concerns ensures that applications with Superficial surfaces are inherently easy to develop, debug and maintain.

Architecture in action – the mechanism

Using the architecture described above, and in particular the capacity of the dynamically constructed targeter tree to retarget the surface, general-purpose sequences can be defined for building and retargeting Superficial surfaces. The sequences below apply to application surfaces; those for dialogs are generally similar.

Building the surface

In the Facets implementation this sequence may be viewed by setting the debug trace flag.

1. A viewable frame is created by a contenter around application content generated from a source such as a file or database connection.
2. For each viewer facet required in the surface, a views target is created which specifies its display policy.
3. Viewer frames are created each wrapping a views target and the shared viewable frame, and from these frames a content area tree to represent the windows and panes that will contain the viewer facets.
4. A suitable layout of viewer and area facets is attached to the content area tree.
5. The content tree is queried for a suitable root targeter, which is retargeted on the root and constructs from it a content targeter tree.
6. The root targeter queries the content area root for the viewer whose facet will have the initial focus, asks its viewable to create a selection tree and from this creates a selection targeter tree. Targeter trees for the active viewer, its views and the viewable itself are constructed in a similar manner and attached to the root targeter.
7. Simple facets and their groupings are constructed, attached to the targeter tree, retargeted and assembled into layouts defining menus and panels.
8. The facet layouts are passed to a GUI host which creates the concrete surface.

Retargeting the surface

In the Facets implementation the results of this sequence can be seen in the debug graph viewer for the surface, which shows the state of the content tree after each retargeting.

1. A facet is notified of input by a widget that it manages, and relays the input to its current target.

2. The target responds by acting on the application element that it represents.
3. The surface root is notified that a retargeting is required (in the Facets implementation by using the targeter tree as a notification tree).
4. The active content root targeter is retargeted on its area root as before, and retargets its child targeter trees; in the Facets implementation any selection target tree is always reconstructed from scratch.
5. Facets are retargeted and update their widgets as required to the state of their current targets.

Because the build and retargeting sequences are so clearly defined, they can be simply encapsulated in template classes with callback methods for application code. This is the approach taken by the Facets implementation.

Coding with Superficial – implementation

The analysis, architecture and mechanism described so far in this paper provide a basis for practical use of the surface metaphor in the design and coding of GUI applications. However they leave open a number of design issues:

- Connecting simple targets to the application
- Initiating retargeting
- Allowing for varying target types

Couplers

The Superficial architecture does not specify how simple targets should be connected to the application elements they represent. The obvious approach is for each target to have callback methods that relay input to the application and define its display and behaviour policy; this however has the disadvantage of increased coupling between surface and application.

To avoid this coupling, each simple target type in the Facets implementation has an associated coupler type enabling definition of application-specific mechanism and policy.

Notification

The Superficial mechanism requires that the surface root be notified when retargeting is required, but does not specify how. As the surface root can be connected via the targeter tree to all target tree members, the Facets implementation defines both targets and targeters as potential members of a notification tree headed by the surface root targeter.

Members of the tree expose a public method to initiate notification, which is usually called by an exposing facet.

Variation in target type

As a result of change in either the selection or the active viewer, a corresponding change may be required in the targeter at a given position in the targeter tree. This is handled as follows in the Facets implementation:

1. Each area targeter and content root targeter maintains an internal list of targeters already created and with attached facets
2. During retargeting, a suitable targeter is either retrieved or created for each target child of the new target.
3. The new targeter tree is attached as necessary to the parent targeter and its attached facets to the surface.

Developing with Superficial – production use

In addition to providing a core implementation of the Superficial architecture as described in this paper, any useful Superficial framework must include completing components that enable practical applications to be designed and coded:

1. The types of the core implementation must be extended and partially specialised, especially those for viewers.
2. Surfaces must be built around targeter trees defined using this extended architecture

Extensions of types from the viewers and applications frameworks in the `facets.superficial` package tree are provided in the Facets implementation by the `facets.app` package tree. The `facets.facet` package tree provides surface building, currently bound to the Swing widget toolkit.

It is worth noting that while these completing components are conceptually fairly straightforward, even in the Facets demonstration implementation they comprise more than four-fifths of the code.

Production use of the Facets implementation in its present state is probably barely practical, but it could be adapted for such use in a number of different ways.

- The simplest route to production use would be by further development of the existing completing components, probably specialised for an application domain. A database application would require specialised viewable and selection frame targets; a graphics application would need to extend the avatars framework.
- The existing somewhat ad-hoc surface builder could be replaced by alternative approaches to defining facets and their widgets and attaching the facets themselves to the targeter tree; in particular, facet layouts could be defined entirely in declarative configuration files.

- Though the core Facets implementation is fairly robust, its API and code are written in the personal idiom of the author rather than with regard to established coding conventions; readability and robustness might be improved by a comprehensive review of the code.

Glossary – Superficial terminology

Because Superficial represents a fresh approach to its problem domain, it unavoidably has its own terminology. Terms defined in this paper are summarised here.

surface	Provides the user of an application with view and control of its content.
view	Enables the user to learn the state of application content; display policy for a viewer.
control	Enables the user to change the state of application content.
selection	Defining a portion of application content to be exposed by the surface; the portion so exposed.
target	Application element to be exposed by the surface; its representation forming part of a target tree.
targetable	Exposes a target directly or indirectly.
facet	Targetable in the abstract surface managing widgets in the concrete surface.
retargeting	Updating targetables with appropriate targets so as to maintain consistency of view and control between surface and application
simple facet	Facet exposing a content primitive or action target.
toggling	Target representing a Boolean value.
numeric	Target representing a number.
textual	Target representing a text value.
indexing	Target representing items and an index into them.
trigger	Target representing an action.
frame	Represents application content while allowing direct access to such content.
viewer	Facet exposing a loose content structure; or target representing such a facet in the area target tree.
avatar	Exposes viewer content to direct view and control

viewable	Content exposed by one or more viewers; or frame around such content and capable of maintaining a selection tree.
selection tree	Represents the current selection.
area	Facet containing either a viewer or other areas, manages a pane or window in the concrete surface; or target representing the facet.
(content) area tree	Represents viewers exposing shared application content.
targeter	Mediates between one or more facets and its own target, forms part of a targeter tree.
(content) root targeter	Targeter for the root of a content area tree, with member targeters targeted on the selection and other target trees.
surface tree	Targeter tree for a complete surface, including one or more content targeter trees
contenter	Provides content area trees exposing its wrapped content and facet layouts for their targeter tree
host	The GUI context for a surface