

within this block of bits describe the many possible differences that exist between diverse specific types of the general/generic Power Supply type. Sample values for a hypothetical power supply were shown in Table 1.

**[0068]** A Power Supply device might have a notion of power control pulse widths (an unlikely notion in other sorts of peripherals). Power On/Off” control is a concept utterly foreign, and inapplicable, to most other types of (peripheral) devices. For instance, sensors (in general) or temperature sensors (in particular) do not typically have any notion of “Power On/Off” control. Neither power-on/off behaviors, nor the associated electrical signal pulse widths, apply to (peripheral) devices in general. In addition, specific power supply sub-types require different power on/off electrical pulse widths.

**[0069]** Power Supply instances can also require different specific wiring choices for the interface to any specific Power Supply. This will depend upon how the Power Supply instance in question is situated within some given system/product. How the system/product is laid out will dictate which host wire/pin must be used for any such Power Supply on/off signal (of whatever width or duration). Which wire must be used for the signal (pulse) is a separate, physical-level matter and is described in the physical scheme.

#### Logical Descriptions Implement a New Type System

**[0070]** In many object-oriented systems, there would be a class to represent each such specific type of Power Supply. Each such class instance could inherit a Power on/off pulse width member (property/attribute) but each class would need different method logic to handle the diverse wiring/interfaces that might be used for any given Power Supply. The present invention makes this unnecessary. Without the present invention, one might (often) need a fast growing number of distinct Power Supply types/classes.

**[0071]** Most other software systems (e.g., object oriented programming languages) generalize based on class/type definitions. This type-based generalization is sensitive to both class member types and class method signatures. A class method signature can include the method’s return value type, the number of method arguments, and the types of each method argument. Using the present invention, declaratively tailored prototypes implement the described facets. Techniques like the block of bits make all facet signatures interchangeable which eliminates the need for most sub-typing.

**[0072]** It is only the interpretation of that block of bits that is sub-type specific (i.e., each sub-type promulgates its own block-of-bits format). This allows for consistent interfaces that work with (and in spite of) different numbers of arguments and different types of arguments. Accordingly, prototypes schemes according to embodiments of the present invention can compose things like a (peripheral) devices schema. This provides a far more flexible sort of processing/programming generalization mechanism.

**[0073]** This more flexible sort of processing/programming generalization permits creation of a fixed software/firmware image that can conditionally incorporate simultaneous support for any number of specific peripheral devices. No new device-specific code needs to be loaded (or linked). Systems using the present invention do not require any sort of reboot, restart or even re-initialization. Even run-time adaptation of the fixed part (to a dynamic hardware environment) becomes quite practical. A few simple changes to the data declared (or added) to the schema (of Peripheral Devices in general) will

suffice. From another perspective, the collection of description scheme instances (i.e., the schema according to embodiments of the present invention) makes peripheral device support fully data driven.

**[0074]** In terms of the running example, a power on/off pulse width of 100 milliseconds might be a common/default value. A longer, or shorter, pulse width might be required by certain specific types of power supplies. The present invention can handle all such variations.

**[0075]** According to embodiments of the present invention, a single, fixed software/firmware image can include a huge variety of very specific type implementations (by brand, make, model, etc.). Some of the fixed logic can manipulate instances of something quite general—like (Peripheral) Devices. Some can manipulate instances of something a bit less general—like Power Supply Devices, while yet other logic in our fixed images can deal with very specific types—like, e.g., a very specific model/device (e.g., “ACME Power Supply model 12A in a FOO model 2500 server computer”).

**[0076]** Due to the super-generalized nature of the schemata (e.g., Power Supply), the present invention offers a unique level of processing/program extensibility. New types can be added both more easily and more powerfully. The schema conventions (i.e., interface contracts) are much less restrictive. Since the individual prototype schemes are more general, new types can more often both conform and inter-operate with existing logic. Existing logic is sensitive only to these much looser prototype scheme interfaces (as promulgated by the more general scheme shapes). The present invention provides an unprecedented level of extensibility (i.e., customizability).

#### Logical Description Artifacts: the “Logical Description Table”

**[0077]** Often, something like a logical description table has a fixed size entry for each instance of any scheme type (e.g., for Power Supply, etc.). A logical description instance/entry represents a scheme instance (e.g., Peripheral Device). The fixed software/firmware image uses the facet interfaces for each listed schema instance (e.g., for each Power Supply scheme instance, etc.). There can be very material (and almost arbitrary) differences between instances of any given scheme type (like Power Supply). Accordingly, much of the described facet machinery is indirect and opaque. Different specific power supply instances can be associated with distinct interface facet sets. In ordinary object oriented programming systems, such different instances would require different Power Supply sub-types. With the prototype based description conventions according to embodiments of the present invention, specific facet schemes can bind to completely different code. In effect, they can bind to sub-type specific implementation logic without requiring any sub-types. The prototype scheme according to embodiments of the present invention is the one and only common thread.

**[0078]** This common thread is kept flexible with mechanisms like the block of bits. All facet signatures within a schema share some arbitrarily unique block of bits. The number of bits is fixed for the entire schema (and thus for each scheme within the schema). Meanwhile, the use/interpretation of these reserved bits is scheme specific.

**[0079]** The interpretation of these bits depends upon the kind of prototype scheme in question. Implementers choose when to declare/define new prototype facet schemes (i.e.,