

Behavioral Set and Field Descriptor Implementations in DPP™

by Chris Brandin

Release 1.1

NOTE: In October 2003, Xpiori, LLC acquired NeoCore Holdings, LLC including all technology and patents. Any references to Neo, NeoCore or NeoCore Holdings, LLC technology or patents as such are now the property of Xpiori, LLC.

**Xpiori, LLC
2864 S. Circle Dr.
Ste. 1200
Colorado Springs, CO 80906
(719) 527-1315
www.xpiori.com**

© 2007 by Xpiori, LLC. All rights reserved.

Version 1.1

Copyright Xpiori, LLC All Rights Reserved

Xpiori technology is protected by the following patents:

US Patent #5,742,611 (21 Apr 98)

US Patent #5,942,002 (8 Aug 99)

US Patent #6,157,617 (5 Dec 00)

US Patent #6,167,400 (26 Dec 00)

US Patent #6,324,636 (27 Nov 01)

US Patent #6,493,813 (10 Dec 02)

US Patent #6,792,428 (14 Sept 04)

Other U.S. and international patents pending.

The information in this white paper has been provided by Xpiori, LLC. To the best knowledge of Xpiori, it contains information concerning the current state of information processing technology. Xpiori, LLC disclaims any and all liabilities for and makes no warranties, expressed or implied, with respect to products described in this paper, including, without limitation, the implied warranties of merchantability and fitness for a particular purpose. No specific reliance should be made on the material provided herein without thorough investigation of the technology and its proposed application to specific circumstances. Product and technology information is subject to change without notice.

Introduction

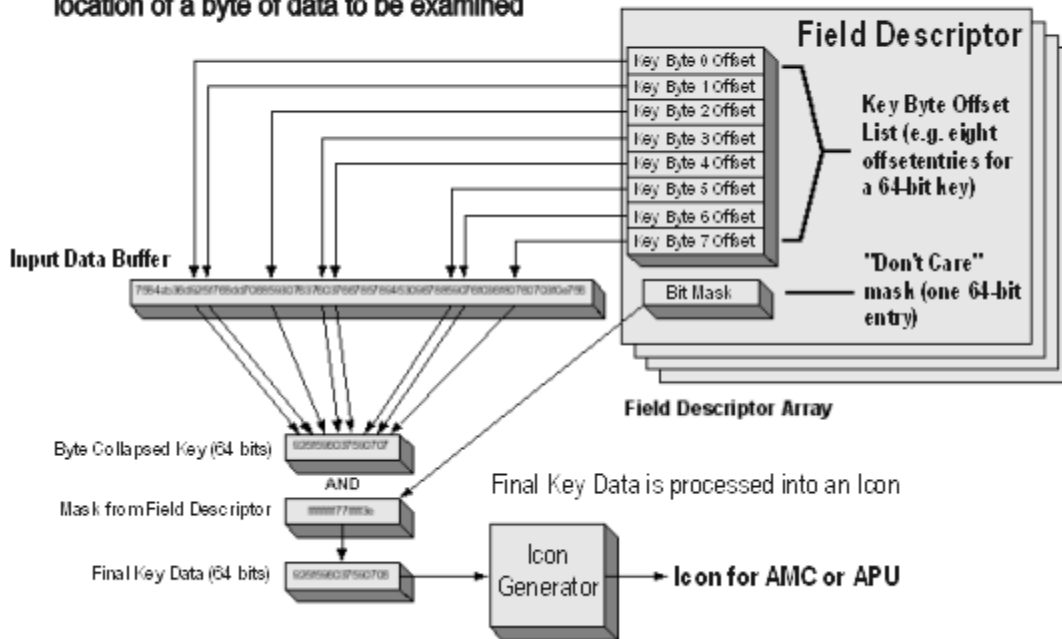
Behavioral Sets expand DPP functionality to include a number of additional capabilities. The most obvious function is the support of predicate logic operations, where an association process includes operations that are contingent on previous ones - for example, looking for a pattern based on the results of a previous match. Behavioral Sets make certain optimizations possible; for example, it may be more efficient to characterize a range of patterns as a "range with exceptions" rather than as a collection. Behavioral Sets also allow a DPP engine to operate based on "what it knows", rather than on a pre-determined algorithm. To support Behavioral Sets, associations are expanded to include additional instructions for the IG/APU. These instructions include references to Field Descriptors (so the engine knows where to fetch additional key data components), and flags indicating which behaviors to invoke (so the engine knows what to do). As the engine associates items, and executes subsequent associations and instructions, its behavior is entirely controlled by the contents of the associations it finds. This makes it possible to build processing structures where a multitude of different processing techniques can be lumped together, and each association "thread" behaves in a way that is optimal for the specific task at hand – or the nature of the data currently being navigated. Although this paper addresses Behavioral Sets primarily, Field Descriptors are described as well because they are critical to making Behavioral Sets work.

New Data Elements

In order to support Behavioral Sets two new items need to be added to the Quanta (association) structure; a Behavior field and a Field Descriptor reference. Also, a new data structure is added – Field Descriptors. When an association is located, and additional processing needs to be done, the Behavior field indicates what should be done next. The Field Descriptor specifies the location of key data that should be used for the next operation. For example, if an association indicates that additional key data needs to be matched, the Field Descriptor would specify where that key data is. A Field Descriptor consists of a list of byte offsets and a mask for "don't care" bits. This allows key data elements to be spread out anywhere in a block of data. A Field Descriptor must be constructed for every pattern of key data that needs to be examined, so an array of multiple Field Descriptors is created. If a Field Descriptor reference is encountered in a Quanta, it is used to select the proper Field Descriptor from the array. DPP engines that use Field Descriptors *always* use them; so a "key" is never presented to the DPP, rather the DPP is instructed to fetch key data (as specified by a Field Descriptor). A diagram illustrating the operation of Field Descriptors follows:

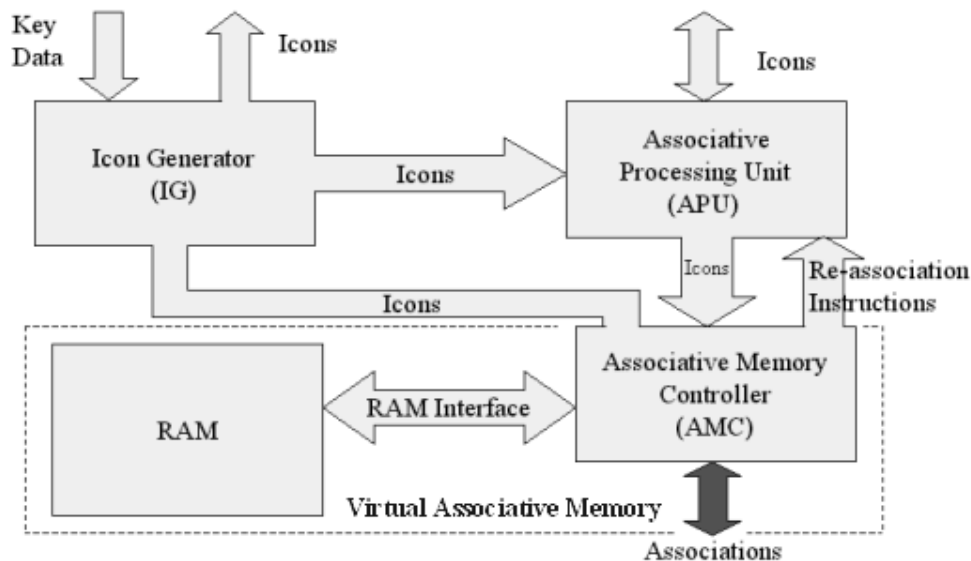
Field Descriptor and Mask Operation

Each Key Byte Offset entry indicates the location of a byte of data to be examined



The Behavioral Set Support Engine

The basic components of a DPP are: the Icon Generator (IG), the Associative Processing Unit (APU), the Associative Memory Controller (AMC), and memory (typically RAM). Below is a diagram of a typical DPP:



When the AMC locates an association, and one or more behavior flags are encountered, the AMC will pass the behavior flags and the Field Descriptor reference back to the APU for processing. The APU will, in turn, cause the new key data, specified by the referenced Field Descriptor, to be Behavioral Set and Field Descriptor Implementations in DPP™

removed. When the thread has completed, the association return value at the bottom of the Association Stack is returned to the user.

The Execution Stack is used to optimize association thread performance. It allows thread execution to continue at a specified Quanta in the event of a "dead end". This happens, for example, if a match condition has multiple exceptions based on different Field Descriptors, and one of the exceptions has an exception to it (an exception to an exception). In this case, execution should continue at the first match conditions' Quanta (not the preceding exceptions' Quanta), in order to look for the next exception.

When an association thread is started, the user specifies a "base set" Field Descriptor to begin with. As the association thread executes, other Field Descriptors are invoked by the Field Descriptor references contained in associated Quantas.

Behavior Types

Behaviors can be established to achieve almost any imaginable logical function. The NeoFilter engine makes liberal use of behavioral sets, so it will be used to illustrate how behavioral sets can be implemented.

The NeoFilter engine was designed to accommodate complex data matching applications, like network packet "drill downs" and fuzzy matching. A number of Behavioral Sets were defined to accomplish this:

- A-Set (Association Set). This indicates that the current Quantas' association value should be pushed on to the Association Stack.
- Q-Set (Qualifier Set). This indicates that additional key data should be iconized as specified in the referenced Field Descriptor, and a subsequent lookup should be attempted. The possible effect of the next association isn't known until it is found.
- T-Set (Test Set). NeoFilter Quantas contain an additional field – "Score". As Quantas are processed within an association thread, the highest running score is maintained. Any Quanta that is subsequently encountered will have its association ignored unless its Score value is higher.
- S-Set (Stack Set). This indicates that if a thread "dead ends", execution should continue at the current Quanta. This is so that if there is another Quanta (a "duplicate"), that is part of the association thread, it can be found (duplicates are linked in a list). Duplicates are defined as Quantas that match the same pattern; their contents can be different.
- E-Set (Exclusion Set). This indicates that the current Quanta represents an exception, so the return association value at the bottom of the Association Stack is removed. There can be "exceptions to exceptions".
- C-Set (Continuation Set). If an A-Set Quanta has been reached via a Q-Set Quanta, the return association value is pushed on to the Association Stack and the association thread is terminated. In some cases this may be undesirable (when the preceding Q-Set quanta has duplicates, for example). If the current Quanta is a C-Set member then the association thread is not terminated, and processing continues. This is an optimization, really. If there were no C-Set behavior, threads would still complete correctly if all Quantas were treated as if they were C-Set Quantas.

A single Quanta can be a member of multiple Behavioral Sets; that is, multiple behaviors can be simultaneously supported. A Quanta can only reference one Field Descriptor, so some logical operations require more than one Quanta. Not all behavior variations are valid. An A-Q-Set Quanta is valid (look for a better match), whereas any S-Set Quanta that is not also a Q-Set Quanta is not valid (there is no subsequent operation to return from).

Using Behavioral Sets

A variety of processing tasks can be accomplished by using Behavioral Sets. A couple of them are discussed below:

- Associating range values. We want to associate any key data value between 5550h and 556Eh with the return association "A", and we also want to associate 9462h – 946Fh with "B". One approach would be to create the following entries:

	Key	Association
1)	5550h	A
2)	5551h	A
3)	5552h	A
4)	5553h	A
5)	5554h	A
6)	5555h	A
7)	5556h	A
8)	5557h	A
9)	5558h	A
10)	5559h	A
11)	555Ah	A
12)	555Bh	A
13)	555Ch	A
14)	555Dh	A
15)	555Eh	A
16)	9462h	B
17)	9463h	B
18)	9464h	B
19)	9465h	B
20)	9466h	B
21)	9467h	B
22)	9468h	B
23)	9469h	B
24)	946Ah	B
25)	946Bh	B
26)	946Ch	B
27)	946Dh	B
28)	946Eh	B
29)	946Fh	B

By using Behavioral Sets the list can be reduced to the following (x indicates "don't care"):

	Key	Association	Quanta Type	Field Descriptor
1)	555x	A	A-Q	2
2)	555F	x	E	x
3)	946x	B	A-Q	2

4)	9460	x	E	x
5)	9561	x	E	x

The Field Descriptors would be:

	Bytes	Mask	
1)	0,1	FFFO	(base set Field Descriptor)
2)	0,1	FFFF	

The ranges are handled by first associating more than the desired ranges and then removing exceptions.

- **Drilling Down.** We want to locate a two-byte data element within a data record, but the data record may contain an optional data field that appears before it. The optional field may be one, or two bytes. The first byte indicates the length of the optional field (0, 1, or 2). The Quanta (A-Set) entries for the desired data elements are created first. In order to find the location of the data element, the following Field Descriptors are created:

	Bytes	Mask	
1)	0	FF	(base set Field Descriptor)
2)	1,2	FFFF	
3)	2,3	FFFF	
4)	3,4	FFFF	

Three additional Quanta entries are created:

	Key	Association	Quanta Type	Field Descriptor
1)	0	x	Q	2
2)	1	x	Q	3
3)	2	x	Q	4

Starting with the base set Field Descriptor, the first byte of a data record is associated. Depending on the byte's value, the associated Q-Set Quanta causes the correct data location (according to the referenced Field Descriptor) to be used for the next association

Complex Searches

For the sake of clarity, very simple examples were cited above. One way to characterize what "Behavioral Sets" brings to the association process is to call it a "non-linear dynamic feedback" mechanism – the same mechanism that drives chaotic, or fractal, systems. As such, Behavioral Sets are capable of accommodating *very* complex association tasks. If applied carelessly, they can also quickly create a mess.

It is usually impractical to manually build data sets for a DPP that uses Behavioral Sets. However, there are many ways to automate the process. These range from using sophisticated logic-minimizing synthesis tools to embedding simple routines tailored to a particular task. For example, the NeoSlider engine is a three-dimensional content scanner that uses a simple Behavioral Set scheme. The entire procedure for creating data sets is encapsulated, so the user never even sees the process.

Trees

Creating search trees is a straightforward process. Parent nodes are implemented using A-Q Set Quantas, and the end nodes (leaves) are A-Set Quantas. Progressive Set Descriptors are used to incorporate increasing amounts of key data for searches. A problem with many applications using trees is that longer matches may be far more common than shorter ones. Trees generally have to navigate through the shorter matches to get to the longer ones. Using Behavioral sets, "inverted" trees can be built. This is because Field Descriptors can be applied in arbitrary order once the initial base set Quanta has been found.

One way to look at Behavioral Sets is as a tree of behaviors. Field Descriptors serve as "templates" for retrieving data. Within a pattern described by a Field Descriptor, any number of association Quantas can be created for various key data values that fit within the template. A Q-Set Quanta can act as a parent node for any number of "child" Quantas. However, any Quanta can also be reached through any number of Q-Set Quantas. So Quantas can have both multiple "children" *and* multiple "parents". Inverted-tree, merged-tree, and lattice data architectures can be implemented.

Conclusion

The use of Behavioral Sets and Field Descriptors brings an entirely new dimension to Associative Processing. DPP makes it possible to address arbitrary and complex data architectures with efficiency. Using these techniques can be a daunting task if attempted manually; however, most of the data manipulation chores necessary to make effective use of Behavioral Sets can be automated.

Appendix

Related Papers

Brandin, Chris, "A Definition of Digital pattern Processing™"

Brandin, Chris, "DPP™ Memory Management"

Brandin, Chris, "Three-Dimensional Content Scanning Using DPP™"

Brandin, Chris, "Optimized Coding Methods for Icon Generation and Manipulation in DPP™"

Brandin, Chris, "Management of Duplicate Data Elements in DPP™ Virtual Associative Memories"

Direen, Harry and Phillips, Keith, "Finite Fields and Properties of the Xpiori Icon Generator, Associative Processing Unit, and Associative Memory Controller used in Digital Pattern Processing™"

#