An application provides two parameters as part of a query: a `summary` and an attribute `label` to attempt to recover from the summary ①. If the summary is a schema summary, the ATN associated with the schema description is used to recover all of the (*label*, *value*) pairs for the attributes stored in the summary ②. Then the value of the requested attribute is returned to the application. If the summary is not a schema summary, then the attribute label is used to probe the structure's underlying Bloomier filter. The retrieved value is then checked against the application-installed false-positive filters ③. The `search` function in the API is effectively a batch operation that applies this same `query` process to a given *label* and a directory of summaries to retrieve the associated value from all of the summaries.

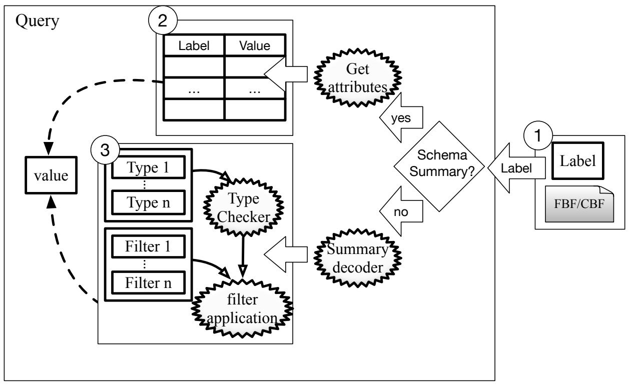

Figure 9: FBF/CBF query process

5.2. The CHITCHAT *Language – The Core*

The CHITCHAT language exposes the key conceptual elements of CHITCHAT so that they can be controlled by applications. Application definitions can leverage other capabilities provided in the CHITCHAT library; the application definitions are in turn interpreted and executed by the CHITCHAT interpreter, which also generates plugins that can be installed on target devices. We describe the core language elements necessary for applications to define CHITCHAT types, false-positive filters, and schema descriptions. The CHITCHAT compiler reads the source code, parses, and generates plugins for user specific applications.

Types. A type in CHITCHAT describes the innate properties of a value in a summary. In CHITCHAT, labels are typed, which means that an application can infer the type of a value from the label associated with the value in a summary. An application can define new types within any of the four groups introduced in Section 2.2: range, encoding, float, and string; applications can also modify existing types. Some examples are shown in Listing 1. Types can be defined as public (i.e., supported as plugins within CHITCHAT) by prepending a '+' to the type definition; a private type, indicated by a '−' is only used in the definition of other types. Line 1 of Listing 1 shows a new type `grade` defined as a range: it is unsigned, has a value between 0 and 100, and is 7 bits. Line 5 shows how the `time` type would be defined; it is an encoding that uses two private types `hour` and `minute`. Line 7 shows a new type of float that specifies a range and shift to enhance false positive detection as described previously. Lines 9 and 10 show examples that define constrained strings: the `event` type allows only alphanumeric characters, while the `name` type should have a length less than 10. Finally, modifying existing types is also possible (Lines 12 and 13). The new type `marketTime` is defined by updating the `hour` of the `time` type.

```
1   +type grade extends range(size=7, min=0, max=100, signed=false)
2
3   -type hour extends range(size=5, min=0, max=23, signed=false)
4   -type minute extends range(size=6, min=0, max=59, signed=false)
5   +type time extends encoding(hour, minute)
6
7   +type temperature extends float(min=-50.0, max=90.0, shift=1.0)
8
9   +type event extends string(alphanum)
10  +type name extends string(length < 10)
11
12  -type marketHour extends hour(min=10, max=18)
13  +type marketTime extends time(marketHour)
```

Listing 1: Type definitions

Filters. Applications that use Chitchat also define false-positive filters that constrain legal values of attributes in context summaries. In Section 2.3, we defined *innate*, *correlated*, and *situational* filters. The *innate* filter is handled by the type definitions. Applications specify the other two types explicitly using the `correlation` and `situation` keywords. Listing 2 gives examples. Line 1 of Listing 2 shows a correlation filter that an application would use to require that `latitude` and `longitude` values must be found together. Correlation definitions can be reused in other definitions; for instance, if a book market application defines a correlation that requires the name, latitude, and longitude of a bookseller, the `location` correlation can be used (Line 2 of Listing 2). The `producePrice` correlation in Line 3 requires that when the summary contains an attribute *produceName*, the summary also has a *price* attribute and its value is consistent with the description. This consistency is defined via a user-specified `priceMatch` function that evaluates whether the *price* of *produceName* is reasonable (e.g., it is reasonable to assume that no piece of produce costs over $1000); we cover the definition of the function later in this section.

While the correlated filters determine whether a context summary is *internally consistent*, i.e., considering only values that are within the given context summary, situational filters are concerned with whether attribute *values* within a summary make sense relative to the receiver's current context. When situational filters check the context summary against the receiver's current status, they may reference local variables (e.g., `here` and `today` in the examples in Listing 2). The situational filter, `within5km`, is only evaluated if the summary contains the attribute `market`, and it requires that the value of an a pair of attributes `(latitude, longitude)` should be within 5 km of the value of some local variable `here`, which references the current location. The situational filter `partyTime` requires that, in a summary that contains a `partyname` attribute, the value of the `(date, time)` attribute should be a time in the future (this filter references the local variable `now`). Finally, Line 7 shows a situational filter, `ageForElementary`, that is evaluated only if the `elementarySchool` attribute is present in the summary (the correlation component). If so, then the `ageOfKid` label should exist and it should have a value between 5 and 12.

```
1  correlation location = (latitude, longitude)
2  correlation bookseller = (name, location)
3  correlation producePrice = (priceMatch(produceName, price))
4
5  situation within5km(market) = |here - (latitude, longitude)| <= 5 _km
6  situation partyTime(partyname) = |(date, time) - now| >= 0 _hour
7  situation ageForElementary(elementarySchool) = 5 <= ageOfKid <= 12
```

Listing 2: Filter definitions

Schemas. In addition to supplementing Chitchat's types and filters, applications can also define schema descriptions to be used to create schema structures. Applications provide these descriptions using the `schema` keyword and these definitions can be either public (exported as a plugin) or private. As above, applications can define both private and public schemas, where the former are simply a syntactic convenience to make defining more sophisticated (public) schemas simpler. Effectively, a schema description extends the capability of correlation filters by adding in the ability to specify some attributes as optional, some with aliases, and to provide repetition of attribute fields. Listing 3 gives examples.

```
-schema sender = (name, id)
-schema datetime = (date, time)
-schema location = (longitude, latitude)

-schema event = event | advertisement
-schema sender = (name, id?) | ("gchat id")

+schema buyQuery = (sender, event, datetime, location)
+schema sensors = (count datetime location? (name id value unit)+)
```

Listing 3: Schema definitions

5.3. The Chitchat *Language – Supplementary Components*

The Chitchat language provides some capabilities that are supplementary to the core but still necessary for properly implementing Chitchat's support for context sharing. In this subsection, we describe these helper elements, which define Chitchat values, functions, and scripts.

Values. The `value` keyword is used to define values; it requires parameters specifying the type(s) of the value(s) used the definition. Chitchat provides predefined values such as `here` (to reference the current location) or `now` (the current date and time). These definitions are shown in Lines 1 and 2 in Listing 4. An application can also define values. A complete example is shown in Lines 4–7 in Listing 4. The `cityParkCenter` value specifies the location of the city park; a situational filter can use the value to check whether a received location is within within 5 km of the park (as shown in Line 9). Note that the `nearCityPark` filter is not correlated to any other attribute in the summary, i.e., the filter has no input parameter; if the situational filter `nearCityPark` is required, and the context summary has both a longitude and latitude value, then the filter will be evaluated.

```
1  value here(latitude, longitude) = // get GPS location as (latitude, longitude)...
2  value now(date, time) = // get current time stamp
3
4  value cityParkCenter(latitude, longitude) = {
5      latitude = [30, 25, 01, 74]
6      longitude = [-97, 47, 21, 83]
7  }
8
9  situation nearCityPark() = |(latitude, longitude) - cityParkCenter| <= 5 _km
```

Listing 4: Values definitions

Commands, Functions, and Scripts. A Chitchat script is a collection of elements specified in the Chitchat language. The script can then be passed to the Chitchat interpreter to be executed by executing the commands in the source code. These commands can (1) call functions (either those defined in the Chitchat APIs or others defined by users); (2) assign the result of evaluating an expression to a variable; or (3) control program flows. A user defined function is indicated by the `function` keyword. A function has a name, input parameters, and a function body. Chitchat users may define functions for a variety of reasons; for instance we saw the `priceMatch` function used in Listing 2 to compare a longitude and latitude to a location representing the city park. This function definition is given in Listing 5. The function depends on the type `cityPark`, which is defined as a point at the center of the city park and a diameter representing the radius of the park. The contains function then checks whether the provided `longitude` and `latitude` are within the specifications of the park.

```
function bool priceMatch(produceName, price) = {
   return ( produceName == "apple" &&  0 <= price <= 1000 )
}
```

Listing 5: Function definition

5.4. Chitchat*'s Execution Models on IoT devices*

We next explain how Chitchat tools can be used to enable context sharing among IoT devices with various capabilities. There are two ways in which an application developer uses the Chitchat tools. First, application developers use the Chitchat APIs to manage the lifecycle of a summary, from creation to serialization for storage and communication. In this use of Chitchat, application developers do not need to interact at all with the Chitchat language and yet can reap the benefits of the compact, space efficient representation, which is essential for IoT communication and storage of context information.