

# Concerns in the Design of a Software Cache

Stanley M. Sutton Jr. and Isabelle Rouvellou

IBM T. J. Watson Research Center  
30 Saw Mill River Road  
Hawthorne, New York 10532, USA  
{suttonsm, rouvellou}@us.ibm.com

## Introduction

Middleware has become increasingly important in the design of modern software systems [2]. As we have argued previously [7], we believe that separation of concerns is an important consideration in both the design and use of middleware. Middleware often succeeds precisely because it supports some needed separation of concerns (such as with language independence in CORBA [6]). On the other hand, middleware often falters when it fails to effectively separate important concerns (as with the language dependence of Sun's JINI technology [9]).

Separation of concerns is also important for middleware as an application domain. Just as the effective separation of concerns enhances the software engineering properties of software in general, so it enhances the software engineering properties of middleware in particular. Since middleware is a fundamental compositional element of many large systems, the ability to maintain, extend, adapt, and reuse those systems may depend critically on the ability to maintain, extend, adapt, and reuse the middleware with which they are composed.

In this paper, we look at concerns in software caches. Software caches are used in many kinds of middleware for performance reasons. Application writers often end up programming their own specialized caches for reasons of performance in particular application contexts. However, a generic, tailorable software cache may be usable in many of these contexts. The ability to adapt, extend, or reuse an existing software cache could save substantially on development times and costs.

In terms of needs and opportunities to adapt, extend, and reuse existing components, a generic software cache exemplifies many kinds of middleware. Caches are also a good experimental subject since, for many purposes, they can be designed as relatively small, self-contained systems while at the same time exhibiting some nontrivial properties. An interesting example is the fact that caches often emphasize nonfunctional concerns and

may put an especially high priority on particular concerns such as performance. The overall combination and relative balance of concerns in caching should raise many interesting issues that we expect to extend to other similarly "unbalanced" middleware services.

The particular cache with which we are working is the GPS cache [3], a "general-purpose software cache" that was developed specifically on the premise that multiple applications ought to be able to reuse basic caching technology. The GPS cache addresses this goal in a number of ways, most notably by being highly configurable. The GPS cache has been used in web server accelerators [5,8] and a business rule manager [1].

The GPS cache is particularly interesting with respect to separation of concerns because it adopts an approach to software adaptability and flexibility that contrasts with that taken by emerging separation-of-concerns (SOC) technologies [4,10]. The GPS cache adopts a strategy of programmed flexibility, whereas these SOC approaches are based on flexible program composition.

The GPS cache also provides an independently developed baseline against which we can compare alternative caches developed using SOC technology. As a first step in this process, we investigated the concerns in the design of the GPS cache. This investigation contributes to a basic understanding of the application domain of software caches. It is required for the development of comparable composition-based caches, and it helps in the formulation of criteria by which alternative caches can be evaluated. Additionally, as we attempt to capture, categorize, and use these concerns, it tests our ability to model important aspects of a software cache across the software life cycle.

In the next section we report the concerns we identified in the GPS cache. We then analyze those concerns, looking at issues related to concern classification, representation, and interrelation. On this basis, we make recommendations for concern modeling. These analyses and recommendations apply specifically to software caches, but we believe that the general points extend to

other kinds of middleware and to software systems in general. Finally, we conclude and indicate future work.

## Concerns Identified in the GPS Cache

The GPS cache was initially implemented in C++ and has been partly rewritten in Java. We identified concerns in the GPS cache based primarily on a description of the design of the cache in its original implementation [N3]. Some of these concerns are explicit in this description; others are implicit. The resulting list of concerns has been supplemented by a few additional concerns that we identified by review of the Java code in the translated version. (In the following tables, concerns marked with an asterisk were added this way.)

Concerns in the GPS cache may be organized in several ways. Here we present them in terms of external versus internal concerns and functional versus nonfunctional concerns. By "external" concerns we mean concerns that are held by, or are apparent to, users of the system. Typically these users will be application developers who are using the cache. However, users may also occasionally include various kinds of software engineers who may work with the cache, such as testers. "Internal" concerns are those that are not (ordinarily) held by or apparent to users of the system. Typically these reflect implementation aspects of the system of interest to developers, maintainers, adapters, and so on.

The specific concerns of the GPS cache are discussed below, along with general characteristics of the cache.

### External Functional Concerns

External functional concerns of the GPS cache are listed in Table 1.<sup>1</sup> Of course, the main function of the cache is to allow users to store and retrieve data. The GPS cache represents data in the form of objects, and it has concerns regarding object addition, deletion, update, and retrieval. Addition optionally allows objects to be inserted into a particular position in a "least recently used" (LRU) list. Retrieval allows for return of references directly to cached objects (faster but unsafe) or to a copies of cached objects (safe but slower).

Invalidation is another characteristic concern of caches, and the GPS cache supports both implicit and explicit invalidation. One reason for implicit invalidation is lack of use; another is violation of inter-object dependencies. Since one of the anticipated functions of the GPS cache

is the storage of interdependent objects, the cache has relatively unusual concerns relating to the addition and

- |  |
|--|
| <ul style="list-style-type: none"> <li>• Add object             <ul style="list-style-type: none"> <li>• With LRU-list position</li> </ul> </li> <li>• Delete object</li> <li>• Update object</li> <li>• Query for object             <ul style="list-style-type: none"> <li>• Returning reference to original (in cache)</li> <li>• Returning reference to copy (ex cache)</li> </ul> </li> <li>• Update object information (e.g., expiration time)</li> <li>• Query for object information</li> <li>• Manage object buffer size</li> <li>• Manage inter-object dependence             <ul style="list-style-type: none"> <li>• Add dependencies</li> <li>• Delete dependencies*</li> </ul> </li> <li>• Invalidate objects             <ul style="list-style-type: none"> <li>• Implicitly (various reasons)</li> <li>• Explicitly</li> </ul> </li> <li>• Control cache logging (turn on, off)</li> <li>• Configure cache parameters (static and/or dynamic)             <ul style="list-style-type: none"> <li>• Memory versus disk storage</li> <li>• Frequency of flushing of transaction log buffer</li> <li>• Maximum object size</li> <li>• Maximum number of objects reclaimable (per GC run)</li> <li>• Expiration interval width (a GC parameter)</li> </ul> </li> <li>• Check for and handle input errors*</li> <li>• Trace execution*</li> </ul> |
|--|

**Table 1: External Functional Concerns**

deletion of inter-object dependencies.

The GPS cache maintains information (meta-data) about cached objects, and it allows users to query this information. The cache also supports the (optional) logging of cache updates and the checking of input.

Various concerns that users may have regarding management and configuration of the cache's structural and operational parameters are also addressed by the GPS cache. Examples of these include maximum object size, total cache size in memory and on disk, and the frequency of flushing of the cache transaction log.

<sup>1</sup> Some of the concerns included here (and in Table 3) might be regarded as "features" rather than "functions", for example, logging and input checking. For simplicity, and to avoid presuming an implementation organization, we group them here.

### External Nonfunctional Concerns

External nonfunctional concerns of the GPS cache are listed in Table 2. These range from typical to atypical.

- Performance
- Data integrity (meta-data and objects)
- Persistence
  - Of cached data
  - Of cache meta-data
- Robustness
- Inter-object consistency
- Information hiding
- Concurrency control (synchronization)\*
- Correctness
- "Generality"

**Table 2: External Nonfunctional Concerns**

Performance is a principle concern for all caches. Data integrity is also a presumed requirement of caches in general. Persistence and robustness are required for many applications. Inter-object consistency is relevant where inter-object dependencies are represented.

Information hiding is a fundamental software engineering concern that is notable here mostly because of ways in which it can be violated. Some implementation features are evident to users. For example, objects can be added by position in the LRU list, and various aspects of the implementation can be configured. Additionally, retrieved objects can be returned as direct references into the cache data store. The GPS cache also allows users to request an object by copy. Information hiding may be violated primarily to support users in the roles of system administrators or deployers rather than end users; however, the exposed information is necessarily visible to all users.

Concurrency control is an issue for caches in general although, for any particular cache, concurrency control may be necessary or not. The original GPS cache (in C++) assumed that users would control concurrency at the application level. Support for concurrency control in the cache was added in the Java translation.

Correctness is a presumed concern of the cache, but it deserves mention because it motivates other concerns.

Finally, in any general-purpose cache, generality itself must be a dimension of concern. Moreover, generality is

a general concern that may have many meanings and may be supported in many ways. Indeed, number of other identified concerns of the cache can be considered to support generality in various ways.

### Internal Functional Concerns

Internal functional concerns of the GPS cache are shown in Table 3. These concerns are recognized functions of the cache implementation.

- Store, delete, update, and retrieve objects
- Manage object information
- Maintain object dependencies
- Log cache operations
- Manage memory-versus-disk use
- Invalidate cached objects
  - By expiration times
  - By data dependence
- Reclaim storage space
- Manage object replacement

**Table 3: Internal Functional Concerns**

Most of these concerns, such as storage of objects and management of object meta-data, are implicit in support of external functional concerns. However, there is not necessarily a one-to-one mapping between the external and internal functional concerns, even when the apparent correspondence is close. For example, adding an object (an external concern) may internally involve storing it (either in memory or on disk), adding corresponding meta-data, possibly logging the addition, and possibly invalidating some not-recently-used object.

Other internal functional concerns are motivated by nonfunctional concerns. Reclaiming storage space contributes to the performance and overall utility of the cache, while managing object replacement helps to assure cache integrity. The tracing of execution contributes to correctness.

### Internal Nonfunctional Concerns

Internal nonfunctional concerns of the GPS cache are listed in Table 4. These apply to elements or behaviors of the cache that are not directly apparent to users.

Many operations on the cache, such as the addition or deletion of an object, will require updates to multiple data structures (e.g., for objects, meta-data, and dependency information). If these operations are terminated prematurely or interleaved with other

operations, then an inconsistent state may result in the cache. Therefore, transactional properties, such as atomicity and isolation, are important for operations on the internal data structures. (Note that operations requiring transactional behavior may occur in response both to external functions, such as adding an object, and to internal functions, such as invalidating an object.)

- Transactional aspects of updates and queries (to objects, object information, inter-object dependencies)
- Internal data structures
- Persistence of internal data structures
- Performance

**Table 4: Internal Nonfunctional Concerns**

In a system such as a software cache, where there is a clear distinction between abstract and implementation structures, the implementation structures themselves become a significant area of concern. For example, an object dependency graph was designed specifically to maintain inter-object dependency information in the GPS cache. Typically, implementation data structures will be selected based on a variety of concerns, often representational appropriateness and (especially for caching) performance at critical tasks.

The persistence of internal data structures is listed as a separate concern since typically the structure of data is orthogonal to its persistence. Persistence of internal structures is required to support persistence of the external structure as well as other nonfunctional concerns such as robustness and data integrity.

The internal performance of the cache of course supports the external performance of the cache. It is listed separately from other internal nonfunctional concerns, such as the internal data structures, because it applies across internal structures and processes both.

### Analysis of Concerns in the GPS Cache.

**Organization of Concerns** There seems to be no one, most appropriate, way to organize the GPS cache concerns. Categorization according to external versus internal and (or) functional versus nonfunctional seems reasonable and natural, but those are not the only possible categorizations. Another categorization would reflect the viewpoints of categories of stakeholders: end users, developers, installers, administrators, and so on. The concerns could also be categorized according to a life-cycle-oriented view, both with respect to when they

are identified and when they are relevant. Still other categorizations might be possible (such as static versus dynamic concerns?). The most appropriate organization of concerns to use in a particular situation depends on the requirements of that situation. This has implications for the organization of artifacts and activities across the software life cycle (for example, in the definition of concern-oriented viewpoints and roles).

**Origin of Concerns in the Life Cycle** The GPS cache was not developed according to a formal life cycle, so we cannot relate the identified concerns to specific life-cycle phases or activities. However, we might hypothesize that the external concerns, which are held by users of the system, would generally be identified during requirements specification, whereas the internal concerns, which are driven largely by implementation issues, would generally be identified during design. The specific concerns we identified seem mostly consistent with this hypothesis, but there are exceptions, especially with the external functional concerns. A number of implementation considerations are reflected in the external concerns. These include, for example, some configuration functions (which specifically configure implementation features), the ability to add objects by LRU-list position, and the availability of "fast" versus "safe" retrieval. Thus, the external functional concerns reflect a composite of requirement-phase and design-phase considerations. The main reason for this, seems to be the high priority placed on the nonfunctional concerns of generality and performance (which the "exposed" implementation concerns support).

**Levels of Concern** The concerns we identified vary in their level of generality. Some, like "add object", are very specific and may correspond to specific, fine-grained implementation units (e.g., methods or variables). Others, like "generality", "configurability", or "performance", are general and may be addressed by a number of more specific concerns. Understanding the interrelationships of concerns, both within and between levels of generality, is critical to understanding how concerns are satisfied. In turn, this can facilitate important software engineering activities across the life cycle, such as requirements verification, design rationalization, dependency tracing, impact analysis, change propagation, and so on.

**Levels of Software** Properties are not solely determined by basic units in a program. Properties may also be determined by (or emergent from) both distinguished subsystems and undistinguished interactions among nominally unrelated elements or features. For example, the performance of the GPS cache overall depends on the efficiency of cache update

operations, on the efficacy of garbage collection in the cache, and on possible interference between cache update operations and garbage collection. In order to represent the full range of influences that program elements may have on various concerns, it must be possible to relate concerns not only to base units but also to organized and unorganized combinations of units.

## Concern Model Recommendations

Because both concerns and software have composite and hierarchical organizations, an appropriate data structure for representing relationships among concerns or software units<sup>2</sup> may be a hypergraph, that is, a graph in which multiple nodes may be grouped into higher-order hypernodes. For example, basic nodes can be used to represent cache update operations and garbage collection, and a hypernode that groups these can be used to represent their interaction. The performance concern can then be made dependent on both the basic nodes the hypernode.<sup>3</sup> A hypergraph could likewise represent relationships among concerns. For example, a hypernode for generality may comprise (hyper) nodes for configurability and for general object types. If a particular concern may depend on a particular module in multiple ways, then it may be appropriate to represent these dependencies using a multi-hypergraph, that is, a (hyper) graph with multiple edges between pairs of (hyper) nodes.

One outstanding issue for the dimensional modeling of concerns (in hypergraphs or otherwise) is how a concern dimension should be coordinatized. This seems especially problematic for nonfunctional concerns. For example, what are the coordinates in dimensions such as performance, robustness, or information hiding? In each case, a variety of answers is possible, but the most useful coordinatization may not be obvious.

Another problem is that the assignment of a language unit to a concern coordinate does not necessarily have a natural or unambiguous meaning. For example, consider the operation to update an object in the cache. This might be done by adding and deleting the object, so it might be less efficient than either of those operations alone. However, it may allow the update to be made with one remote procedure call instead of two, and therefore improve overall performance. So, with respect to the performance concern, we have a question as to which coordinate the update operation should be added and also the question of how the assignment to that

coordinate should be interpreted. Note that other units might be added to the same performance coordinate but for different reasons. For example, garbage collection also improves performance, but not in the same way as an efficient update operation or an efficient update protocol. Thus, we believe that mappings of software units to concern dimensions and coordinates may often need to be accompanied by semantic annotations.

## Conclusion and Future Work

The design of the GPS cache involves a number of functional and nonfunctional concerns which can be further divided into external and internal concerns (among other categories). The importance of nonfunctional concerns, especially generality and performance, makes the design of the cache implementation critical and leads to the mingling of internal and external concerns (strongly implying concern entanglement in the implementation).

Concerns identified for the GPS cache originate from several sources. The structure of the concern space is interesting in that it exhibits composite concerns and concerns on multiple levels. These have relationships to program elements with a correspondingly complex structure. These structures and relationships may be best captured by a (possibly multi-) hypergraph. Beyond concern structures, there remain issues in the capture of nonfunctional concerns and dependencies.

We plan to formalize further the model suggested in this paper as a basis for creating the design and implementation for a “composed” version of the GPS cache. This composed cache can then be compared to the existing “programmed” version, both with respect to basic cache features, functions, and performance, and with respect to properties such as adaptability, extensibility, and reusability. Of special interest will be the ability to accommodate new and additional concerns, especially those such as distribution, concurrency, and events, that are particularly important for middleware and middleware mediated systems.

## Acknowledgments

We thank Arun Iyengar and Lou Degenaro, both of IBM T. J. Watson, for their advice regarding the original GPS cache and the Java translation, respectively.

---

<sup>2</sup> Which may be from a programming language, design language, or any other kind of language used in software development.

<sup>3</sup> In terms of Hyper/J [10], the performance hypermodule would be composed of (roughly) the update, garbage collection, and combined update plus garbage collection units.

## References

- [1] Degenaro, Louis, Iyengar, Arun, Lipkind, Ilya, and Rouvellou, Isabelle. *A Middleware System Which Intelligently Caches Query Results*, in Joseph Sventek and Geoffrey Coulson (eds.), Proceedings IFIP/ACM International Conference on Distributed Systems Platforms, New York, NY, USA April 2000 (Middleware 2000); Lecture Notes in Computer Science 1795, Springer-Verlag, Berlin, pp. 24-44, April, 2000.
- [2] Emmerich, W. Software Engineering and Middleware: A Roadmap. In: Finkelstein, A. (ed): *Future of Software Engineering – State of the Art Reports given at the 22nd Int. Conf. on Software Engineering*, Limerick, Ireland, 2000. ACM Press, 119-129.
- [3] Iyengar, Arun. *Design and Performance of a General Purpose Software Cache*; in Proceedings of the 18th IEEE International Performance, Computing, and Communications Conference (IPCCC'99), Phoenix/Scottsdale, Arizona, February 1999.
- [4] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. M., and Irwin, J. *Aspect-oriented Programming*. In *11th European Conference on Object-Oriented Programming*, pages 220–242. Springer-Verlag, June 1997.
- [5] Levy, Eric, Iyengar, Arun, Song, Junehwa, and Dias, Daniel. *Design and Performance of a Web Server Accelerator*, in Proceedings of IEEE INFOCOM '99, New York, New York, 1999.
- [6] OMG. The Common Object Request Broker: Architecture and Specification, rev. 2.2.OMG, 1998.
- [7] Rouvellou, Isabelle, Sutton Jr., Stanley M., and Tai, Stefan. *Multidimensional Separation of Concerns in Middleware*; in Workshop on Multi-Dimensional Separation of Concerns in Software Engineering -- Proceedings (22nd International Conference on Software Engineering), 2000, pp. 106 - 111.
- [8] Junehwa Song, Levy, Eric, Iyengar, Arun, and Dias, Daniel. *Design Alternatives for Scalable Web Server Accelerators*, in Proceedings of the 2000 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2000), Austin, Texas, April 2000.
- [9] Sun Microsystems, <http://www.sun.com/jini/>.
- [10] Tarr, Peri, Ossher, Harold, Harrison, William, and Sutton Jr., Stanley M. *N degrees of Separation: Multidimensional Separation of Concerns*, in Proceedings of the 21st International Conference on Software Engineering, ACM, New York, 1999, pp. 107--119.