System architecture

Design, Architecture

We suspect that the real reason is the lack of a comprehensive, hysteretic and unifying approach to architectural design that makes the patterns In some sense comparable. Acquirement specification into a working software and hardware system and, hence, could be seen as " programming-in-the-very-large". Since it is an accepted doctrine that mistakes when caught in the early stages are much cheaper to correct than when discovered in the late stages, good architectural system design could be of enormous economical potential. The purpose of this paper is to take a first step in the direction of a methodology for architectural design. Or in other words, we submit that architectural design should allow a methodology and not intuition, I. E. Should be treated as ascienceand not as an art. In order not to become overly ambitious, and to stay within the confines of a conference paper, we will limit ourselves to information systems as the synthesis of data base and datacommunicationsystems, with more emphasis on the former. 2 2. 1 Services Services and resources Since we claim that architectural design is the first step in a process that turns a requirements specification into a working software and hardware system, an essential ingredient of the design method is a uniform and rigorous requirements specification.

Requirements is something imposed by an outside wow RL. For information systems the outside world are the business processes in some real-world organization such as industry, government, education, financial institutions, for which they provide the informational support. Figure 1 illustrates the basic idea. The counterpart of business processes in an information system are informational processes. Business processes proceed in a linear (as in Figure 1) or non-linear order of steps, and so do the informational processes.

To meet its obligations, each step draws on a number of resources. Resources are infrastructural means that are not died to any particular process or business but support a broad spectrum of these and can be shared, perhaps concurrently, by a large number of processes. In an information system the resources are informational in nature. Because of their central role, resources must be managed properly to achieve the desired systemgoalsof economy, scale, capacity and timeliness.

Therefore, access to each resource is through a resource manager. In the remainder we use the term information systems in the narrower sense of a collection of informational resources and their managers. What qualifies as a resource depends on the scope of a process. For example, in decision processes the resources may be computational such as statistical packages, data warehouses or data mining algorithms. These may in turn draw on more generic resources such as database systems and data communication systems.

Business Informational process 1 Process step 1 Resource manager 1 Process step 2 Process step 3 Resource manager 2 Process step 4 Resource manager 3 Process step 5 Resource manager 4 process 2 Figure 1 Business processes, informational processes and resources What is of interest from an outside perspective is the kind of support a resource may provide. Abstractly speaking, a resource may be characterized by its competence . Competence manifests itself as the range of tasks that the resource manager is capable of performing.

The range of tasks is referred to as a service. In this view, a resource manager is referred to as a service provider (or server for short) and each subsystem that makes use of a resource manager as a service client (or client for short). 2. 2 Service characteristics The relationship between a client and a server is governed by a service level characteristics of the services it provides. From the viewpoint of the client the server as to meet certain obligations or responsibilities. The responsibilities can be broadly classified into two categories.

The first category is service functionality and covers the collection of functions available to a client and given by their syntactical interfaces (signatures) and their semantic effects. The semantic effects often reflect the interrelationships between the functions due to a shared state. Functionality is what a client basically is interested in. The second category covers the qualities of service. These are non-functional properties that are nonetheless considered essential for the usefulness of a server to client. 2. Service qualities To make the discussion more targeted, we study what technical equal ties of service we come to expect from an information system. Ubiquity. In general, an information system includes a large - in the Internet even unbounded - number of service providers. Access to services should be unrestricted in time and space, that is, anytime between any places. Ubiquity of information services makes data communication an indispensable part of information systems. Durability. Information services have not only to do with deriving new information from older information but also act as a kind of business memory.

Access to older information in the form of stored data must remain possible at any time into an unlimited future, unless and until the data is explicitly overwritten. Durability of information makes database management a second indispensable ingredient of information systems. Interpretability. In an information system, data is exchanged across both, space due to ubiquity and time due to durability. Data carries information, but it is not information by itself. To exchange information, the sender has to encode its information as data, and the receiver reconstructs the information by interpreting the data.

Any exchange should ensure, to the extent possible, that the interpretations of sender and receiver agree, that is, that meaning is preserved in space and time. This requires some common conventions, e. G. , a formal framework for interpretation. Because information systems and theirenvironmentusually are only loosely coupled, the formal framework can only reflect something like a best effort. Best-effort interpretability is often called (semantic) consistency. Robustness. The service must remain reliable, I. E. Guarantee its functionality and qualities to any client, under all circumstances, be they errors, disruptions, failures, incursions, interferences. Robustness must always be founded on afailuremodel. There may be different models for different causes. For example, a service function must reach a defined state in case of failure (failure resilience), service functions muss t only interact in predefined ways if they access the same resource (conflict resilience), and the effect of a function must not be lost once the function came to a Security.

Services must remain trustworthy, that is, show no effects beyond the guaranteed functionality and qualities, and include only the predetermined clients, n the face of failures, errors or malicious attacks. Performance. Services must be rendered with adequate technical performance at given cost. From a client's perspective the performance manifests itself as the response time. From a whole community of clients the performance is measured as throughput. Scalability. Modern information systems are open systems in the number of both, clients and servers.

Services must not deteriorate in functionality and qualities in the face of a continuous growth of service requests from clients or other servers. 3 Service hierarchies 3. 1 Divide-and-conquer Given a requirements specification in terms of service functionality and qualities on the one hand and a set of available basic, e. G. , physical resources from which to construct them on the other hand, architectural design is about solving the complex task of bridging the gap between the two.

The time -proven method for doing so is divide-and conquer which recursively derives from a given task a set of more limited tasks that can be combined to realize the original task. However, this is little more than an abstract principle that still leaves open the strategy that governs the decomposition. Higher-levelresponsibilityarrive functionality qualities composition: assemble higher-level responsibility decomposition: divide higher-level lower-level responsibilities Figure 2 Divide-and-conquer for services We look for a strategy that is well-suited to our servicephilosophy.

Among the various strategies covered in [Est.] the one to fit the service philosophy best is the assignment of responsibilities. In decomposing a larger task new smaller tasks are defined, that circumscribe narrower responsibilities within the original responsibility (Figure 2). If we follow Section 2. 2, a responsibility no matter what its range is always fined in terms of a service functionality and a set of service qualities. Hence, the decomposition results in a hierarchy of responsibilities, I. E. Services, starting from the semantically richest though least detailed service at the root and progressing downwards to ever narrower but more detailed services. The inner nodes of the hierarchy can be interpreted as resource managers that act as both, service providers and service clients. 3. 2 Design hypothesis All we know at this point is that decomposition follows a strategy of dividing responsibilities for services. Services encompass functionality and a large number of laity-of-service (So) parameters. This opens up a large design space at each step.

A design method deserves its name only if we impose a certain discipline that restricts the design space at each step. The challenge now is to find a discipline that both, explains common existing architectural patterns, and systematically constructs new patterns if novel requirements arise. We claim that the service perspective has remained largely unexplored so that any discipline based on it is as yet little more than a design hypothesis. Our method divides each step from one level to the next into three parts. Functional decomposition. This is the traditional approach.

We consider service functionality a a primary s criterion for decomposition. Since the original service requirements reflect the needs of the business world, the natural inclination is to use a pure top-down or stepwise must decide whether, and if so how, the functionality should be further broken up into a set of less powerful obligations and corresponding service functionalities to which some tasks can be delegated, and how these are to be combined to obtain the original functionality. However, the closer we come to the basic resources the more hose will restrict our freedom of design.

Consequently, at some point we may have to reverse the direction and use stepwise composition to construct a more powerful functionality from simpler functionalities. Propagation of service qualities. Consider two successive levels in the hierarchy and an assignment of So- parameters to the higher-level service, we now determine which service qualities should be taken care of by the services on the upper and lower levels. Three options exist for each quality. Under exclusive control the higher-level service takes sole responsibility, I. E. , does not propagate the quality any further.

Under partial control it shares the responsibility with some lower-level service, I. E. , passes some So aspects along. Under complete delegation the higher-level service ignores the quality altogether and entirely passes it further down to a lower-level service. For partial control or complete delegation our hope is that the various qualities passed down are orthogonal and hence can be assigned to separate and largely independent resource managers. Priority of service qualities. Among the service qualities under

exclusive or partial control, choose one as the primary quality and refine the decomposition.

Our hope is that the remaining qualities exert no or only minor influences on this level, I. E. , are orthogonal to the primary quality and thus can be taken care of separately. Clearly, there are interdependencies between the three parts so that we should expect to iterate through them. 4 4. 1 Testing the design hypothesis Classical 5-layer architecture Even though it is difficult to discern from the complex architecture of today's relational DB'S, most of them started out with an architecture that took as its reference the well-published 5-layer architecture of System R [Sass, Chic].

Up to hose days the architecture is still the backbone ofacademiccourses in database system implementation (see, e. G. , [HERR]). As a first test we examine whether our design hypothesis could retroactively explain this (centralized) architecture. 4. 1. 1 Priority on performance We assume that the DB'S offers all the service qualities of Section 2. 3 safe ubiquity, the relational data model in its SQL appearance. As noted in Section 2. 3, durability is the raisin d'©tree for DB'S. Durability is first of all a quality that must be guaranteed on the level of physical resources, by non- volatile storage.

Let's assume that durability is delegated all the way down to this level. Even after decades durability is still served almost exclusively by magnetic disk storage. If we use processor speed as the yardstick, the overwhelming bottleneck, by six orders of magnitude, is access latency, which is composed of the movement of the mechanical access mechanism for reaching a

cylinder and the rotational delay until the desired data block appears under the read/write head. Consequently, performance dwarfs all other service qualities in importance on the lowest level.

Considering the size of the bottleneck and the fact that performance is also an issue or the clients, it seems to make sense to work from the hypothesis that performance is the highest-priority quality across the entire hierarchy to be constructed. 4. 1 . 2 Playing off functionality versus performance Since we ignore for the time being all service qualities except performance, our design hypothesis becomes somewhat simplified: There is a single top-priority quality, and because it pervades the entire hierarchy it is implemented by partial control.

The challenge, then, is to find for each level a suitable benchmark against which to evaluate performance. Such a benchmark is given by an access profile, that is a sequence of operations that reflects, e. G. , average behavior or high-priority requests. We refer to such a benchmark as data staging. More expressive data model data staging data model Id wider usage context access profile resource manager I less expressive narrower Figure 3 Balancing functionality and performance on a level Consequently, our main objective on each level is determining a balance of functionality and data staging.

As Figure 3 illustrates, the balancing takes account of a tandem of knowledge. On the way down we move from more to less expressive data models and at the same time from a wider context, I. E. More global knowledge of prospective data usage, to a narrower context with more

localized knowledge of data usage. The higher we are in the hierarchy, the earlier can we predict the need for a data element. Design for performance, then, means to put the predictions to good use. Based on these abstractions we are indeed able to explain the classical architecture. We start with the root whose functionality is given by the relational model and SQL. The logical database structure in the form of relations is imposed by the clients. We also assume an access profile in terms of a history of operations on the logical database. We compress the access profile into an access density that expresses the probability of Joint use of data elements within a given time interval. The topmost resource manager can now use the access density to rearrange the data elements into sets of Jointly accessible elements.

It then takes account of performance by translating queries against the relational database to those against the rearranged, internal database. The data model on this internal level could very well still be relational. But since we have to move to a less expressive data model, we leave only he structure relational but employ duple operators rather than set operators. Consequently, the topmost resource manager also implements the relational operators by programs on sets of tepees.

What is missing from the access density is the dynamics - which operations are applied to which data elements and in which order. Therefore, for the next lower level we compress the access profile into an access pattern that reflects the frequency and temporal distribution of the operations on data elements. There is a large number of so-called physical data structures tailored to different patterns - or combined associative and sequential

access. The resource manager on this level accounts for performance by assigning suitable physical structures to the sets of the internal data model.

The data model on the next lower level provides a library of physical data structures together with the operators for accessing them. It is not all clear how to continue from here on downwards because we have extracted all we could from the access profile. Hence we elect to change direction and start from the bottom. Given the storage devices we use physical file management as provided by operating systems. We choose a block-oriented file organization because it makes the least assumptions about subsequent use of the data and offers a homogeneous view on all devices.

We use parameter settings to influence performance. The parameters concern, among others, file size and dynamic growth, block size, block placement, block addressing (virtual or physical). To lay the foundation for data staging we would like to control physical proximity: adjacent block numbering should be equivalent to minimal latency on sequential, or (in case of RAID) parallel access. The data model is defined by classical file management functions. The next upper level recognizes the fact that on the higher levels data staging is in terms of sets of records.

It introduces its own version of sets, namely segments. These are defined on pages with a size equal to block size. Performance is controlled by the strategy that places pages in blocks. Particularly critical to performance is the assumption that record size is much lower than page size so that a page contains a fairly large number of records. Hence, under the best of circumstances a page transfer into main memory results in the transfer of a

large number of Jointly used cords. Buffer management gives shared records a much better chance to survive in main memory.

The data model on this level is terms of sets of pages and operators on these. This leaves Just the gap to be closed between sets of records as they manifest themselves in the physical data structures, and sets of pages. Given a page, all records on the page can be accessed with main memory speed. Since each data structure reflects a particular pattern of record operations, we translate the pattern into a strategy for placing Jointly used records on the same page (record clustering). The physical data resource manager places or retrieves records on or from pages, respectively.