A Zoned Architecture for Large-Scale Evolution

Ray Welland
Department of Computing Science
University of Glasgow
Glasgow, G12 8QQ, UK
+44 141 330 4968
ray@dcs.gla.ac.uk

Malcolm Atkinson
Department of Computing Science
University of Glasgow
Glasgow, G12 8QQ, UK
+44 141 330 4359
mpa@dcs.gla.ac.uk

ABSTRACT

This position paper describes our notion of *zones* to support the incremental evolution of persistent application systems. We focus on the motivation for our work and the basic concepts underlying our zoned architecture (ZEST).

Keywords

Software architecture, zones, evolution, persistent application systems.

1. INTRODUCTION

The focus of our work is the evolution of large-scale and long-lived application systems which we refer to as Persistent Application Systems (PAS) [1]. Such systems are inherently complex and heterogeneous, and in order to support the incremental evolution of a PAS we introduce the notion of *zones*. A zone is a logical subdivision of a PAS, independent of physical constructs, and will normally correspond to a partition of the management structure of the organisation which the PAS supports. We expect zones to evolve largely autonomously except at the critical boundaries where they exchange information. We believe that the identification and description of zones makes the evolution of a PAS a more tractable problem because of the focus on a small number of critical features, abstracting away from much of the implementation detail.

An architecture, ZEST (Zoned Evolvable Software Technology), is introduced that helps software engineers to manage and achieve evolutionary maintenance steps in a PAS. ZEST focuses on the interactions between zones. It provides mechanisms for describing these interfaces, generating code for the transmission of data between zones, and supporting the incremental evolution of these interfaces. In this paper we will discuss the motivation for our work, introduce the basic concepts of ZEST, and the philosophy underpinning it. We will draw our examples from a health care system that inspired us to propose this architecture.

1.1 Motivation

The dominant activity in the software industry is maintenance. In very large systems it is impractical for software engineers to develop a knowledge of the whole system in order to accomplish changes in a safe and timely manner, yet it is still essential that new components or changed components operate correctly in the existing context. Using ZEST we seek to identify the relevant logical structure within a PAS and to automate the insertion of adaptive components that limit the propagation of the effects of a change.

ZEST focuses attention on a *partial* description of a PAS. It is infeasible to describe every aspect of a typical PAS, but it is realistic to incrementally describe the aspects of the system that are relevant for a particular evolutionary step. Cumulatively, that process will describe more and more of a PAS and hence the context of future changes, but only at the coarse grain that is the concern of ZEST and only for the information flows that the system maintainers consider relevant. We postulate a notional role, *PAS architect*, who describes this structure, though in reality it may often be performed incrementally by the members of teams who design and implement changes.

We therefore arrive at three requirements for the ZEST architecture; it should:

- cope with the typical properties of a PAS, namely scale and heterogeneity;
- be applicable when changes are being made to existing systems; and
- be definable incrementally while still providing benefits when it contains only a partial description of a PAS.

1.2 Basic concepts

The ZEST architecture is an independent description of a PAS that is explicitly constructed by software engineers, i.e. it is not derived automatically from software components. In order to accommodate heterogeneity, to support description of planned subsystems and to focus on critical properties, automatic derivation is precluded.

ZEST describes a partitioning of the PAS into *zones* and a definition of the information flow between zones. This information flow is defined in terms of *gateways*. A zone is a logical partition of a system, which should reflect the managerial structure of the organisation, which the PAS supports. A zone is independent of physical constructs, such as particular processors, machines, databases,

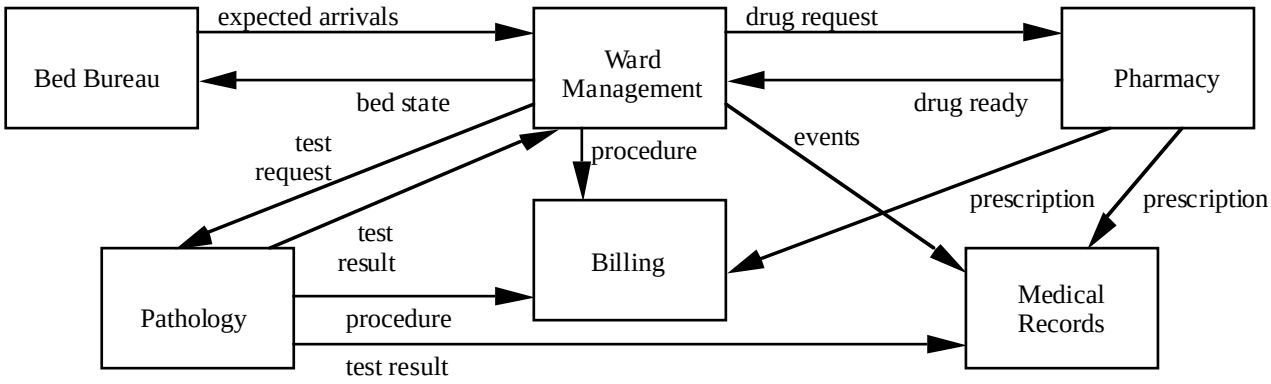

Figure 1. Some Zones from the HMS

languages, etc. A zone is typically large, i.e. much larger than software modules, and may contain many programs, processes, databases, or views of databases. The internal structure of a zone is not the concern of ZEST.

ZEST describes certain external properties of zones, particularly their interaction with other zones. Information that is the concern of ZEST flows from one zone to another via a gateway. Each gateway defines a unidirectional contract, which determines the information the source zone is committed to supply to the receiver. Similarly, it constrains the information flow to only the data that complies with this contract. It is these constraints that prevent the *described* structure of the PAS from decaying. Each gateway is unidirectional because the information that may flow from zone A to zone B is frequently different from that permitted in the inverse direction, B to A.

2 Motivational Example

The authors observed the evolution of a health care management system over the period of five years as members of its design team. The basic concepts of ZEST were considered significant during this work, but without an articulated and supported architecture they were unsustainable.

2.1 The HMS experience

The Healthcare Management System (HMS) was intended to provide *integrated* support for all activities in a health district's hospitals, their ancillary services and the surrounding medical practices. It is the goal of integration over such a broad enterprise that necessitates new architectures. Previously, the support for: a pathology lab, the pharmacy, the ward, the operating theatre, medical records, etc., were each quite separate. These units were accustomed to autonomy, in particular to choosing and tailoring their own IT systems which were large but tractable. This independence, however, had obvious drawbacks, such as a growing number of terminals and systems for staff to cope with, and consistency and labour costs incurred as information is manually transferred between systems.

Faced with the need to integrate many systems, and to cope with the required diversity and autonomy, an implicit architecture was proposed. We will describe it in terms of ZEST nomenclature, although that has developed more recently.

The intention was that the whole system would be comprised of largely independent zones. These zones would correspond to well-defined activities within the healthcare system and would have limited communication via a number of distributed relational databases. The zones would each have a schema defining their data, which might cohabit on various databases with the schema of other zones. Communication was to be restricted by requiring that data was moved from one zone to another by pre-packaged queries. These parametric retrievals and updates were *intended* to be the *only* knowledge one zone had of another. The schemas were intended to remain independent.

Zones were progressively commissioned (i.e. the clients committed to their construction), developed and introduced. The partitioning however was not sustained. Under the pressure of concurrent development of the operational zones and new work, the software team progressively interlaced the zones via direct and unplanned (or even erroneous) use of other zone's information. This led to a rapid growth in complexity that undermined the potential of the project. Hence the idea of ZEST was born.

2.2 Example Health Care System

As a running example we will consider a few zones of a simplified HMS, shown in fig. 1. This is only a small sample of the 20 to 30 zones. For example, `Outpatients` generates a structure similar to `Ward Management` and `Bed Bureau`. All of these zones inter-work with `X-ray`, `Physiotherapy`, `Dieticians`, `Clinicians`, etc. There are zones for `Clinical Management` and `Nursing Management`, which interact with `Payroll`.

These zones hold large volumes of data and many programs (50 relations, 100 programs and 130 screen definitions, in the case of `Ward Management`, for example.), only *a small proportion* of which is of concern to other zones. Although there are potentially n^2 gateways (where n is the number of zones) allowing information flow between zones, the actual connectivity is *a sparse subset* of this.

The important information recorded in a ZEST architecture identifies and delimits these "small proportions of data in a zone that are of external interest" and "this sparse subset of permitted and useful data flows". Although they may be relatively small, the total structure of a PAS that is of interest at the ZEST level is so large and complex that it needs organising, is normally constructed incrementally, and must be exploited systematically. This organisation, incremental construction and systematic use is the essential role of ZEST that was missing in the HMS project.

3 PHILOSOPHY and PRINCIPLES

3.1 Separability of Roles and Responsibilities

Examination of human organisations, e.g. industrial and commercial enterprises, governmental organisations, educational and health care establishments, etc. shows that it is managerially essential to partition their activities. We observe that this means that there are commitments about communication between these divisions that are relatively stable, e.g. `Estates and Buildings` must tell `Bed Bureau` and `Nursing Management` which wards are in service and what beds they contain. The organisational structure would be unworkable if this inter-departmental communication were not relatively stable and a small part of each department's work.

Every PAS serves a human organisation, and its structure should reflect and support this partitioning of an organisation. Once this happened naturally as individual decisions were made about commissioning and changing IT systems. *With the advent of enterprise-wide integration this sympathetic structure must be identified and utilised deliberately*. We believe that it will prove relatively stable (departments resist changes to their roles and working practices) and beneficial if it is communicated by the PAS architects to the PAS maintainers. If zones comply with this logical structure most change occurs within a zone and the ZEST definitions ensure that commitments to other zones are sustained. When changes in commitments are required the gateway definitions provide the foundation for negotiating a new contract. The revised transport and translation software is then generated automatically.

3.2 Heterogeneity & Legacy Systems

Each zone may be developed using a variety of technologies. These technologies will vary because different technologies may be appropriate for different applications or because the software and databases in a given zone were commissioned when different technologies were popular. In consequence, the descriptions in ZEST should be independent of the technologies used in a zone, but must describe a mapping to them. To do this for every aspect of a zone and for every technology would be hopelessly uneconomical. However, as ZEST only concerns itself with the external interactions of a zone and the *relatively* few data structures that are transmitted through gateways this is tractable.

The externally visible structures of each zone, the data flow through gateways and data translations are defined in terms of a high-level type system. Currently this is based on the Collection Programming Language (CPL) [2]. We believe that such a high-level notation, properly supported by tools, will be comprehensible to software engineers, will support inter-working over a wide variety of communication protocols between all of the programming languages and databases that may be encountered. There is good evidence that such notations will encompass inter-working with virtually all legacy systems.

ZEST's strategy for legacy systems is to describe just those parts of the legacy system that interact with other zones of interest. The primitive communication operations required by ZEST involve intermediary software that emulates a zone with the specified gateways while operating the legacy system's proprietary interfaces.

Organisations change their choice of supporting technologies from time to time. By explicitly describing the mapping to these technologies, ZEST is able to generate new software while maintaining the APIs, for example when moving between RMI (the Java inter-process communication protocol) and versions of CORBA.

3.3 Incremental Evolution & Construction

PAS are rarely built in a "green field site". The nearest approximation to this comes when a new subsystem is commissioned. The largest unit commissioned will be as much as the organisation can afford to install in one event. It is straightforward to describe the new zone. What is more challenging is to describe and circumscribe its interaction with existing surrounding systems. In the HMS system described above each of the zones was commissioned separately.

Just as organisations commission new systems in units, which are small enough to manage, they also commission changes with affordable cost and limited disruption. Changes occur at two levels: minor-maintenance in which a subdivision of the organisation commissions a change that it perceives as solely its concern, and major maintenance where the change requires co-operative agreements from a group of interacting subdivisions. As an example of a minor change, `Ward Management` would re-use beds while patients were in theatre or intensive care. To do this they allocated additional lockers for the extra patients. Tracking whose belongings were in which locker and where the locker was, was an entirely parochial issue. When `Ward Management` started to record "responsible person" details, as well as "next of kin", this affected `Medical Records` as well as `Bed Bureau` and the information sent on a `Patient Admission` event. It was therefore major maintenance.

3.4 Control and Autonomy in Evolution

It is essential that a software architecture deliver its benefits strongly in the context of change. The thrust of ZEST is to allow PAS architects to specify information flow in such a way that the benefits for change management are significant. The effects are different under the two forms of change described above. For minor maintenance, we anticipate that the change will take place within one zone. The description of this zone's gateway commitments will

then allow the required internal changes to be made, but insist that translations are introduced if necessary to restore the commitments. Normally most of the translations are generated automatically.

When a major maintenance change is underway, it may be decomposed into several minor maintenance operations and some which require new forms of information flow. The ZEST descriptions then identify which commitments need to be renegotiated and re-specified and encourages the specification of any totally new commitments. The overall goal is to localise as much change as possible, so that software engineers can perform local changes autonomously while relying on a stable environment, despite the contemporaneous changes being developed by others elsewhere in the PAS. Only where it is necessary is this autonomy restricted by commitments to other zones that the PAS architects have chosen to specify.

3.5 Incrementally Taming Chaos

It is assumed that introduction of the ZEST approach would normally take place in the context of a system that is already operational. That is, there would be a large collection of programs, databases, communication protocols, etc. already operational. These would involve a whole variety of technologies: languages, databases, middleware, scripting systems, etc. It would be quite infeasible to attempt to describe the whole of such a system before beginning maintenance with the assistance of ZEST. Accordingly, we imagine that only the parts of the existing system that are of immediate interest will be named and described.

As successive increments and changes occur, further structure will be described, so that gradually more of the maintenance activity can be supported. It is therefore essential that ZEST does not assume that it has a total model. There will be "covert channels" and unknown parts.

A structural model will exist in the mind of anyone planning a change *for that part of a PAS involved in the change*. As this change is planned and moves through implementation and commissioning, we would expect the description of this understood structure to be captured. It is then available to manage this change and all subsequent evolution of the system that interacts with the described parts. Progressively more of the system is thereby described.

4 CONCLUSIONS and FURTHER WORK

An architecture has been proposed that describes a carefully selected high-level structure commonly apparent in the large scale software applications that service the needs of an enterprise. It is believed that this structure is particularly helpful in supporting an application system's evolution. The essence of the architecture is the division of the system into coherent logical zones that correspond to organisational roles and the specification of the limited information flow between these zones. The pay-off is the partially automated maintenance and constraint of those information flows.

4.1 ZEST: the Story So Far

A preliminary version of ZEST has been built and small-scale evaluation has been undertaken [3]. Currently, the part of a zone that is visible to ZEST must be coded in either C++ or Java and the communication must be via CORBA or RMI. We have automatically generated the gateway code when different communication technologies are installed and when different languages are used at each side of the gateway. Similarly we have shown how to automatically restore commitments when the exportable types supplied or expected by a zone are changed. A study with only three zones was inconclusive, but we believe that the promise of ZEST would be fulfilled in a larger-scale evaluation.

4.2 Evaluation on Realistic Projects

The next step should be a realistic evaluation. This is difficult to organise because of the labour involved in experimenting with a large PAS, because of the commitment and access that is required from the owners of a PAS and because the effects that must be observed would only manifest during an experiment of reasonable duration. Extensions in the technologies supported and the tool set would probably be necessary. In particular communication via databases, as in HMS, would need to be supported and tools which helped with the generation of mappings to common technologies, or at least their validation would be extremely helpful.

5. ACKNOWLEDGEMENTS

This research was supported by EPSRC Research Grant GR/K79598. The experimental version of ZEST was implemented by Cathy Waite and David Jack. We would like to thank Graham Technology Ltd for their co-operation in the field trial.

6. REFERENCES

[1] Atkinson, M.P. and Morrison, M. Orthogonally Persistent Object Stores, VLDB Journal, 4, 3, 1995.

[2] Buneman, O.P., Libkin, L., Suciu, D., Tannen, V. and Wong, L., Comprehension Syntax, ACM SIGMOD Record, 23, 1, 1996.

[3] Waite, C., Welland, R.C. and Atkinson, M.P., Supporting Software Evolution in ZEST, Technical Report TR-1997-29. Department of Computing Science, University of Glasgow, 1997.