Integration of Campus Wide Information systems using a hub and spoke architecture.

Alan Berg, Jan Bode, Joost Bataille

Central Computing Services, Universiteit van Amsterdam, The Netherlands.
a.m.berg@uva.nl, J.A.Bode@uva.nl, J.R.L.M.Bataille@uva.nl

Abstract

The Universiteit van Amsterdam (UvA) [12] is geographically displaced throughout the millennium old capital city. As a reflection of this and of the devolved powers of the organization there exists's a diversity of information systems and solutions scattered physically, architecturally and administratively. To lower cost of maintenance and to approach, in the near future, a more uniform and consistent biosphere of systems a data integration project was initiated. This project involved the transition from a number of diverse structures glued together with scripts with many to many relations to the use of a well known architectural pattern, the 'hub and spoke' model via an industrial standard product Oracle Interconnect [10]. The product enabled the possibility of near real time synchronizations. This paper details the ins and outs and highlights potential profits and indeed the potential pit falls.

Keywords: Integration, Java, IMS.

1 Introduction

The Universiteit van Amsterdam has a long and renowned history, being established in 1632 as Athenaeum Illustre. The population at present comprises 25,000 students and 5,000 members of staff, 55% of which are academics. Having close links with the city of Amsterdam is an advantage. However this indirectly implies that the campus itself is highly distributed with roots in many diverse physical locations. The underlying technical infrastructure had also reflected this scaterlogical reality. The Mieloso[1] project was initiated to alleviate this complexity of inner plumbing. Figure 1 pictorially explains what happens when you allow infrastructure to organically over time grow without weeding.

The disadvantages of this situation are:

(1) Many to many relations

If there are numerous interrelationships then the number of failure paths can only severely increase. Sometimes these failure paths are obvious and sometimes the critical path is hidden under the snow of detail. Maintenance and debugging costs escalate as a polynomial with each extra node added.

(2) Biodiversity of applications

Biodiversity has a tendency to cost money. The more applications, the more application specific knowledge is required to maintain to the in-house baseline standard. The natural implication of which implies more consultancy hours, training, documentation and other types of effort.

(3) Diverse administrative zones

Interface boundaries are the natural place for miscommunication. The diversity of zones increases the total length of these boundaries.

(4) Reliability issues

The more paths the greater the opportunity for a significant failure or bottlenecks, race conditions, errors in code etc. Inconsistencies in work practices add also to the background noise.

(5) Scripting is often said to be "write once and be confused often"

Perl, bash, csh, awk, sed etc. represent sources of instant code satisfaction. However the code tends to be personalized to the given administrator involved and grows in the passing of time into a potential burden. For example, there has been more than one occasion that one of the authors has written a complex reglex expression to crunch a particular input from a text file that fails x months later on an unexpected entity. The author later returning to review the reglex and at that moment being unable to quickly interpret the reglex's clever yet flawed inner workings.

(6) Overview and monitoring is difficult

The old adage "What you do not measure, you do not control" is valid. Diverse systems require diverse monitoring solutions. Not all the systems produce counters readable by a useful industrial standard

[1] A project initiated by the Development Sector of the Informatersingscentruim to retain insight and thus control over provisioning between diverse systems.

protocol such as SNMP. [3] Therefore custom code is normally the end result of the process and is almost a requirement for smoothing out the diversity of the underlying network management protocols that each application may expect.

(7) Inertia towards implementing new technical realities

Technology improves, things change and chances occur, new conventions and practices become standard. Often this is not easy to place in a diffuse structure. Certain questions are more difficult to answer. What if we break something? Which systems are affected? Uncertainty breads resistance.

(7) Security

The more diffuse the relationships between systems the more opportunity for crackers. The greater the number of lines of glue codes the more opportunity for crackers. The strongest security is limited by its weakest link.

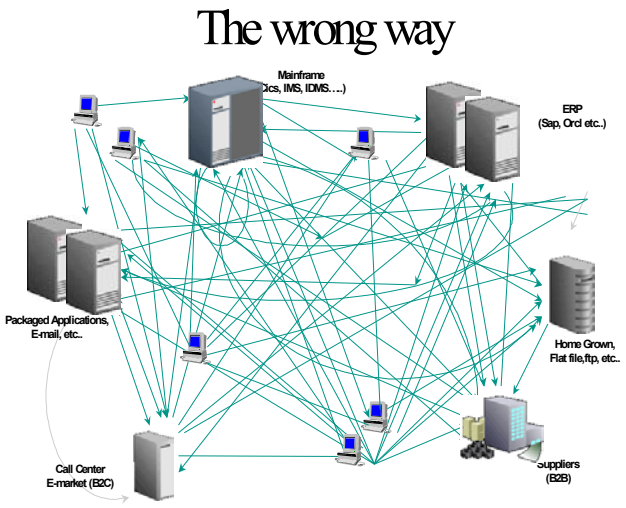

Figure 1: An example of a non planned organically growing infrastructure.

However, shown in figure 1, after all is said and done there is one significant advantage of such a fuzzy approach as. The growth occurs naturally especially if you do not actively plan. This growth is the governance equivalent of not weeding your garden.

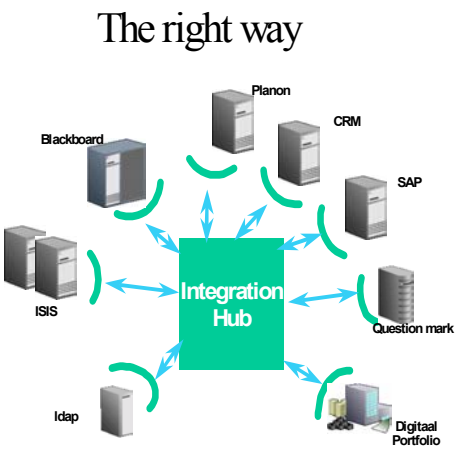

Figure 2: Hub and spoke integration.

The Mieloso project placed the Oracle Interconnect product at the centre between an increasing numbers of applications. Information that is filled in the administration, is sent to the hub via an adapter. The adapter sends IMS compliant XML [6] that is processed by the hub and then passed on to other applications via other instances of adapters. The hub is responsible for the workflow and is a central point of monitoring and control. The adapters are composite entities that can be divided into two: The first part is the queuing mechanism that ensures stability. The second part is the end point of the message that does the actual work. If a message fails to get to the hub or from the other side to the application the adapter can perform two primary actions; the hub can ignore the issue or repeat sending the message after a random period of time. This message queuing adds an extra degree of stability. Within the UvA circumstance stability is quite important. An observable pressure on the infrastructure is related to work practices; every Monday evening there is a campus wide maintenance slot where the system administration crews are free to install updates, replacements, patching etc. During this slot the uptime of all systems are not always 100% guaranteed. If a router fails and the network is down then the adapter will keep trying to deliver its guaranteed message until the misbehaving subsystem is replaced. Monday evening blues have little effect now on data synchronization.

Summarizing the properties of the interconnect adoption:

- Hub and spoke model
  the hub controlling the flow between applications and the spokes being the adapters that queue, translate and send messages and convert to native events.
- Relatively low latency
  Message passing from start point to finish node if

not interrupted can be measured under normal circumstances in seconds.

- Trivial Load distribution
  it is possible to block the next request to an adapter until the last message has finished delivering its payload correctly. This allows for load distribution flattening.
- Reliability through message queuing
  if a connection fails, a message may be queued or ignored according to the logic of the situation.
- A Number of standard adapters
  HTTP, FTP, Advanced Queue exists. Note, in reality there will be in most common deployment scenarios much custom coding
- Java SDK for building your own adapters. The development sector here at UvA has standardized their production effort around the Java programming language. Therefore Java API's are quite helpful and part of the buy in decision of any project.
- Extra adapters for an extra price.
  Adapters for Peoplesoft and SAP products are available. The SAP adapter being of potential interest due to the nature of our main administration systems
- Kiss methodology.
  Keep It Simple Stupid; the team used a standard HTTPS adapter instead of building on the generic adapter. All custom coding was done from the applications side cleanly separating responsibility between configuration of the infrastructure and application specific code.

2 10 km view

In this section the overall process is described and in the following section the details of synchronization between the student administration and Blackboard [5] are zoomed into so as to give you an idea of the flavour of a 'typical' implementation of an adapter.

The Mieloso project in phase one integrated the student administration with Blackboard and LDAP [4] and indirectly through LDAP the mail system for students and an Active Directory forest.

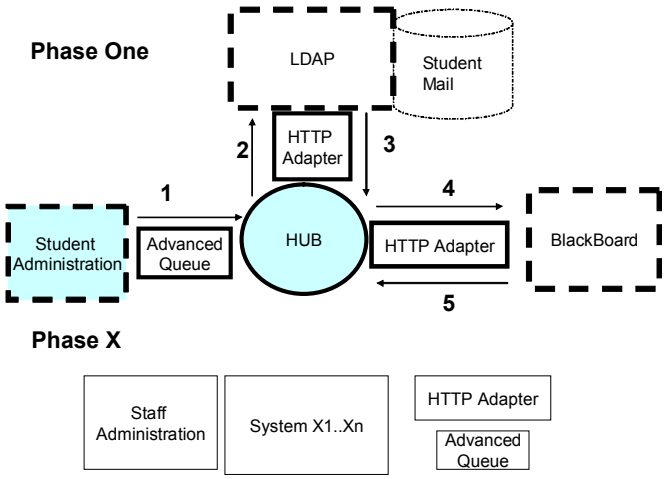

Figure 3: A simplified hub and spoke integration for LDAP, student mail and Blackboard. Phase X shows the pieces of the jigsaw required to add extra applications and data drivers.

Figure 3 shows the sequence of events for creating a new user.

(1) An event, in this case a user creation event, triggers a message being sent via an advanced queue adapter to the hub.

(2) The message is processed and sent through an HTTP adapter to a servlet. The format of the message is XML in a binary stream. The XML conforms to the IMS enterprise schema. This standardizing increases the opportunity for future compatibility.

(3) The servlet updates LDAP and then sends the e-mail address back of the newly created record. in an updated IMS XML message.

(4) The hub sends the e-mail address as a field in the modified IMS XML message to a custom Java Server Page (JSP) in Blackboard.

(5) The JSP through a relatively complex series of events makes the user. A 200 OK response is then sent back and the message is removed from the queue. If an error occurs, than depending upon its value the adapter can decide to repeat sending the message or ignore the issue.

An HTTPS adapter [8] was found to be a safe solid bet. The adapter can send messages via both the HTTP 1.0 and HTTP 1.1 protocol standards [7], using HTTP POST and GET operations. The messages are sent in the form of an XML binary stream. To reply to the adapter is a matter of sending a status code back. Within the project it was decided to attach to applications via only this one type of adapter and thus allow for homogenous configuration and testing. At no point was the decision to simplify regretted.

The most important design compromise was the use of XML in IMS (Enterprise) format [9]. This implied an extra programming effort to write a custom XML parser, but in return promised the most likely possibility later of out of the box interoperability with other applications. Further adding extra applications and data sources to the hub would be much easier in any following phase of the project, the parser being reusable for future joining. This is directly due to the reliance on the HTTPS adapter for the applications and the advanced queue adapter for the main administration data resources. Most of the configuration would be the same and only minor nudges would thus be necessary. Of course custom coding would still occur natively, but in well defined locations and when Java is involved a common code base can be enforced and reused by an experienced and battle hardened team.

3 The Blackboard adapter

Blackboard as a pilot started in the year 2000. Now the online learning environment is the major e-learning environment at the University. About 90% of the regular university courses have an online component. Further, Blackboard is coupled with LDAP directory Services via Perl scripts for record synchronization. Qua usage is the system at times a little busy, especially between the hours of 11am-4pm. In the third quarter of 2004 there were 431,213 visits with an average session length of around 9 minutes comprising 36,000,000 hits. This translates visually to figure 4. The figure shows unique content browsed per hour against date. The lower peaks are in the weekends.

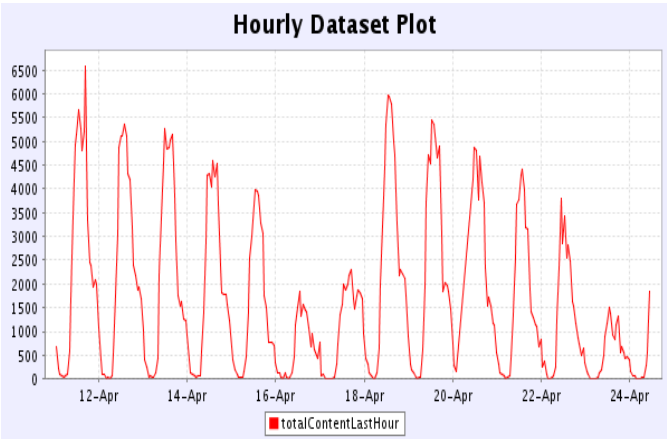

Figure 4: Unique content hits per hour vs. date in April 2005 as measured by a UvA specific management system [1].

Counter Type	*Value*
Number of active users [2]	14,450
Courses enabled	3,375
Average enrolment per course for all courses.	62.50
Number of enrolments in courses	**248,000**
Groups	4,200
Average group size	12.80
Group memberships	**53,750**
N° of content files known to system	153,000

Table 1 Snapshot of UvA Blackboard counters as of April 24th 2005.

The new infrastructure needs to be able to deal with at least the historical volume of change and a potential bursty nature at the beginning of semesters. Table 1 outlines relevant counters. From the table it can clearly be seen that if ever the database needs to be filled from scratch then the total product of events needed to create a course is dominated by the 250,000 associated course enrolments.

Blackboard has a snapshot tool for data, that can import either comma separated or xml formatted text files for record synchronization. This tool has in Holland previously been used in a number of data integration projects []. This command line tool is based on the Java based Blackboard data integration API. After careful consideration and discussion it was decided to use the same API to achieve our predefined goals. Blackboard is proprietary software; however the integration API is openly published. This API was extensively applied via writing a wrapper of UvA specific libraries. The libraries performed XML parsing, workflow, marshalling and logging and managed all the little details to our in-house style. The libraries were imported and called through a thin layer of logic encapsulated in JSP. Later, when the message is sent from the HTTPS adapter it calls the JSP. The JSP checks whether the call is valid, in this case against an IP address manager. Next the page calls the underlying wrapper libraries and through them the data integration API layer. The libraries perform the required actions against java objects that sit in the same Java Virtual Machine (JVM) as part of the Blackboard servlet or directly against the underlying database. Depending on the outcome the JSP simply returns a status code. This is clearly shown in figure 5.

[2] This is measured by a date mentioned in the last login field of the accumulator table. This table is purged to a statistics database once a month and thus obviously defines the users logged in during the last month.

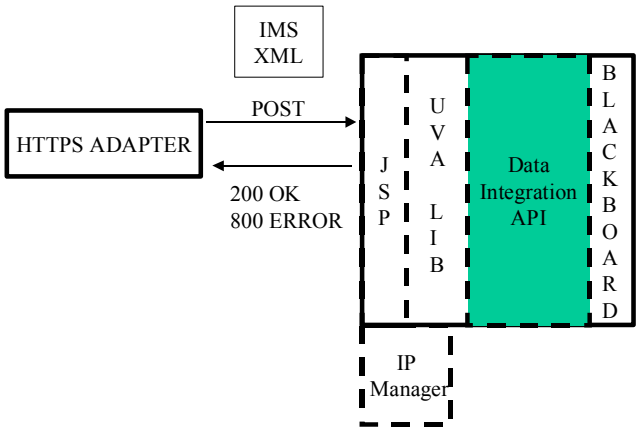

Figure 5: Simplified diagram of the design of the Blackboard synchronization.

The customized part of the adapter enabled the following functionality:

Creation, deletion, modification

(1) Person(s)
New persons were created with a random password. The password was not directly required as Blackboard authenticated against LDAP. The random password ensured that no one could hijack the account in a fallback situation when the LDAP binding failed.

(2) Course(s)
A new course is created if a course does not exist that is similar from the year before. If the course exists then certain parts of the course are copied including course documents and instructor enrollments, but not student enrollments. Yes, this turned out to be a tricky piece of code. The course copying was achieved via a proximity search on name and relied heavily on the SQL like statement to find the older course names. Please note that this part of the code is tender and may change over the course of time.

(3) Course enrollment(s)
A user may exist once in Blackboard, but also per course that the user is enrolled in. The reason for this is that a student has per course entitlements that are properties of a given instance of the enrollment.

(4) Workgroup(s)
Workgroups can exist under courses. Members of a workgroup may share a group's email address, discussion board, virtual classroom and also a common file sharing mechanism.

(5) Workgroup membership(s)
A workgroup membership is first a member of a course and then a member of a group. The overall entitlements are a combination of all these instances.

From the previous list it soon becomes obvious that there is an order of operation to messages passing from student administration to the application. A course needs to be made, then the students and instructors need to be enrolled and then the groups need to be generated and finally the enrolled member in the course needs also to be enrolled in the group. The supply chain can be quite long. Out of order processing would logically fail. Consistency was enforced by applying well defined business logic via constraints within the data input application of the student administration.

4 Lessons Learnt

The Mieloso project turned out to have a considerable learning curve associated with it. The configuration of the Oracle Interconnect product was detailed and securing the system required consideration in every phase and every pour of each phase. Each extra application required specific extra knowledge. The leaving of an audit trail by the hub and by the adapters was helpful generating a reasonable overview and solid base for planning and debugging. In general the system has proven itself robust and reliable not giving us as developers any heartburn. IMS XML parsing cost us extra effort, but has measurable benefits in the future. The use of XML in IMS enterprise format gives us confidence of relevance in the near to middle term future for the HTTPS adapter and also as an escape route for applications that now read this format but may be split off from Oracle Interconnect when necessary as the technological/political landscape changes later.

From the team programmer perspective within this project Java was a good language for maintenance. Code is split into classes with easily definable responsibilities. The use of UML [11] especially the Class and sequence diagram gives a pictorially intuitive understanding of the overall relationships in the design. The Blackboard Java API was comprehensive enough for our requirement. However, there was unexpectedly a too steep learning curve for the API. In fact because of the lack of documentation and code examples, much trial and error was required at the time of the development window in the project.

The adapters and holding servlet containers required much tweaking, but once configured remained stable and predictable.

On the subject of reusability: once the overall structure was in place and running it was found to be easier to expand than a long series of Perl scripts. The HTTPS adapter allows for a deliberate monoculture of similarities. Much of the code is effectively already written for any future addition to the

infrastructure. The XML parsing can be replicated and the basic servlet structure is common to a majority of modern applications.

The negatives: one must state that you must look carefully at the range of adapters available. The range probably will not reach to every part of your requirements. Be prepared for much custom coding despite the softly flouted allure of standardization and do not think that the adapters will just birth straight out of the box and fit magically within your environment. Look carefully at other solutions especially identity management Metadirectory products [2]. Where do you want to keep your identity information? Have you a campus wide uid that needs to be consolidated? This project was specifically not about identity more about general governance issues.

Keep design simple. For example the XML parsing within the native part of the adapter was designed to understand multiple objects in XML, but later that was found not to be required.

Finally, keeping the system up and tuned may very well depend on many details so spread knowledge within the organization.

5 Summary and Conclusions

From the discussion in this article it can be seen that it is viable to replace diverse communication relationships with a hub and spoke architecture using an industrial standard product. The deployment itself has had no significant issues. The one noticeable negative is that the initial learning cost was high, but does tail off over phased iterations. The reuse of Java based code and adapter configuration will dramatically lower development efforts in future iterations.

Future proofing any product is difficult but IMS XML message passing no matter which product uses it is a solid basis for such an attempt. Oversight and control of our technological process driven backyard has been refined, planning made easier and adaptation to new situations simplified. The many benefits out weigh the initial costs and yes it was a fun project.

Acknowledgements

The team would like to acknowledge the support and detailed feedback given by Marc van den Berg and look forward to the continuation of these joint efforts.

References

[1] A. Berg, V. Maijer, F. Benneker. “Blackboard 6 usage patterns and implications for the Universiteit van Amsterdam”, *Eunis proceedings 2003.*

[2] B. Belling. “Architecting your data and Metadirectory model”. http://www.educause.edu/ir/library/powerpoint/nmd0310.pps

[3] D. Levi, P. Meyer “RFC 3413: Simple Network Management Protocol (SNMP) Applications”. http://www.isi.edu/in-notes/rfc3413.txt

[4] M. Wahl, T. Howes, S. Kille.“RFC 2251: Lightweight Directory Access Protocol”. http://www.ietf.org/rfc/rfc2251.txt

[5] Blackboard. http://www.blackboard.com

[6] Extensible Markup Lanaguage XML. http://www.w3.org/XML/

[7] HTTP protocol. http://www.w3.org/Protocols/

[8] HTTPS adapter. http://www.oracle.com/technology/products/integration/htdocs/httpsadapter.html

[9] IMS Enterprise Best Practice and Implementation Guide. http://www.imsglobal.org/enterprise/enbest01.html

[10] Oracle Interconnect. http://www.oracle.com/technology/products/integration/htdocs/interconnect902ds.htm

[11] Unified Modelling Language UML. http://www.uml.org/

[12] Universiteit van Amsterdam. http://www.uva.nl http://www.uva.nl/organisatie/object.cfm