A New Architecture

Evolution of DPIWE e-business applications

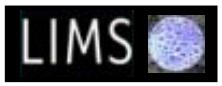
Kade Hansson

Overview

In the last five years, DPIWE has developed a number of web-based business applications. They range from systems which are almost completely accessible to the public to those which are used entirely by people within the department.

The systems selected for examination in this discussion paper are mainly chosen for the unique characteristics of their deployment mode and underlying technologies. They are:

- 

Land Information System Tasmania (LIST)
  - Publicly accessible system accessible from modern web browsers with some dependencies on proprietary technologies. The LIST is a critical portal for DPIWE's Land Services division.
  - Initially envisioned as a prototype system, this system is still in wide use today with no immediate plans of replacement.
  - Based on a variety of technologies, but primarily Oracle PL/SQL.
  - Code is mainly ad hoc and there is little possibility of reuse.
- 

Laboratory Information Management System (LIMS)
  - Internal business system with a sound internal design, but suffering from a mismatch between product and client needs. Dependent on Internet Explorer with Java and JavaScript enabled.
  - While the architectural design is sound, the implementation is largely quite poor and highly inefficient. In particular, LIMS would benefit from application of the MVC[1] design pattern.
  - An intricate mosaic of technologies which is hard to develop and maintain, placing strain on organisaional expertise as a result. Includes a large number of Java servlets, a modest number of Java applets and JavaScript scripting on the client side.
  - LIMS uses Java Database Connectivity (JDBC) directly without any architectural abstraction, and the complexity of the code reflects this. Documentation is virtually non-existent.

[1] Model View Controller. A common Java 2 Enterprise Edition (J2EE) design pattern where data and business logic are separated from presentation form.

- STARS

  Service Tasmania Automated Receipting System (STARS)

  - Internal business system dependent on a Java technology enabled web browser. STARS meets the clients' needs quite well.
  - Built around a successful proprietary object-oriented/relational hybrid architecture produced by ALife Consulting called JALife. Newer versions of the source code for this architecture are not open to inspection or development from DPIWE's perspective, except in conjunction with STARS.
  - Based on JDBC technology, but predates J2EE. Uses an applet on the client to achieve the maximum potential for delivering usability and productivity for the clients.
  - Code is of a fairly high standard, but is quite complex, not very well documented, and therefore difficult to maintain. Reuse is possible, but with limitations due to the JALife license.

- SurCoM

  Survey Control Marks Database (SurCoM)

  - Internal business system with a clever tiered approach to database access. Runs on all browsers with very few dependencies on proprietary technologies. SurCoM is reasonably successful, and is going to be released to the Australian Antarctic Division as the basis of their marks database.
  - LIMS in a bottle, but the tiered approach to database access is almost J2EE EJB BMP[2], yet SurCoM predates this technology. SurCoM benefits from the application of the MVC design pattern.
  - Uses the same JDBC engine as LIMS, but this has been branched in SurCoM and is not shared. SurCoM uses JSPs[3] for content delivery, but ignores some of the newer and more advanced features, essentially resulting in what are really "ASPs in Java."
  - Code is of a modest standard, but is very poorly documented in places. In particular the JSPs are almost unreadable, and fail to separate content from semantics and presentation elements (save for the MVC structure, which lifts SurCoM to the level of being almost manageable.)

[2] Enterprise Java Beans with Bean Managed Persistence. An older J2EE development model where business entities are mapped onto objects which must provide their own database access code. Typically there is one bean per table in the database.

[3] Java Server Pages. A J2EE technology for producing mark-up language content on the server side. Similar to Microsoft's Active Server Pages (ASPs), but with better facilities for separating the concerns of software developer and layout designer.

- Land Titles System

  Tasmanian Folio and Valuation Information System (Tasfol/Vistas)

  - Like STARS, Tasfol and Vistas are internal business systems dependent on a Java technology enabled web browser. Vistas may allow WAP access via the (yet to be developed) Ms Architecture extension. Tasfol/Vistas is still under development, but is widely regarded as DPIWE's best web application to date in terms of client satisfaction.
  - Uses the object-oriented/relational architecture Mr Architecture which is slated for an open source release. Developed internally by Kade Hansson, a DPIWE employee, the emphasis in Mr Architecture is on the an object-oriented approach which can leverage modern technologies in a lightweight and reusable framework that is efficient in terms of both development and execution.
  - Mr Architecture is very close to J2EE EJB CMP[4], and future evolution to Dr Architecture will likely see a complete J2EE compliant EJB container produced with the possibility for migration of Tasfol/Vistas with some effort.
  - Code is of a high quality, and well documented. The code supporting the architecture is highly complex when compared with other DPIWE systems (except JALife perhaps), but the good level of documentation mitigates maintenance difficulties somewhat. Some of the Tasfol business logic is bound to UI components due to Mr Architecture providing limited support for business transactions which cross bean boundaries.

Even from this brief analysis, it is clear where the weak points lie. And as a production system can only be as good as the architecture which supports it, and development methodology which created it, we must therefore reexamine these two areas before we embark on new projects or we are doomed to repeat the same mistakes.

It is worth pointing out that these analyses can only be made in hindsight. It is extremely difficult to see, and even more diffciult to change, a development architecture or methodology that is failing a current project. The criticisms above are made only to allow DPIWE to move forward, and do not reflect on the talented people who produced the systems. Without their efforts, there would be no basis for improvement, as the CIT division would have been shut down long ago.

[4] Enterprise Java Beans with Container Managed Persistence. A recent advance in J2EE technology which allows database access code to be automatically generated by the supporting container implementation. The developer provides business objects which map onto database tables in a way declared by **deployment descriptors**.

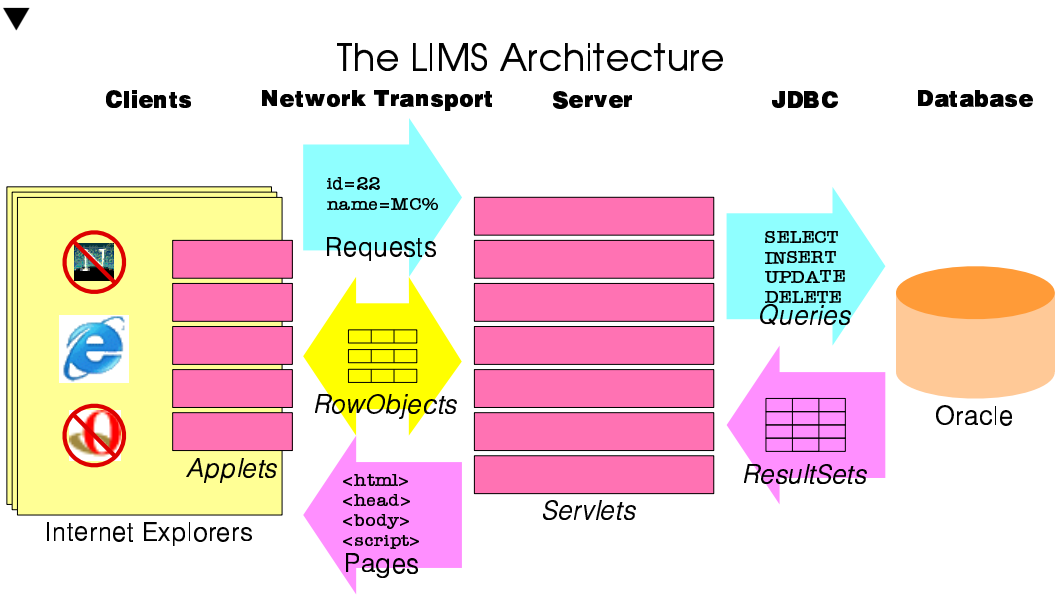
One positive that is worth noting is that all of these projects resulted in a working production system with a level of resources that would be considered impossibly low in most organisations. This fact alone is a tribute to all developers, past and present, who have had a hand in creating the present range of DPIWE business information systems.

The Current Architectures

We will ignore the LIST's architecture, as it is a production system only by accident. It is a successful experiment in producing a web application, but its development owes more to the talented people that worked on it than the mainly ad hoc architecture which supports it.

LIMS and STARS were almost parallel developments, with LIMS the youngest and least successful of the two:

- LIMS' architecture abstracts JDBC into persistent *RowObject*s containing data objects which encapsulate the data fields in each row, but nothing more. There is one object type for each table (like EJB). There is also one servlet for each object type which serves *RowObject*s and accepts data objects for insertion or update. All of the presentation layer is coded by hand, with one servlet per possible client request. Database access code (save complex queries) is produced automatically by the GeoCASE tool.

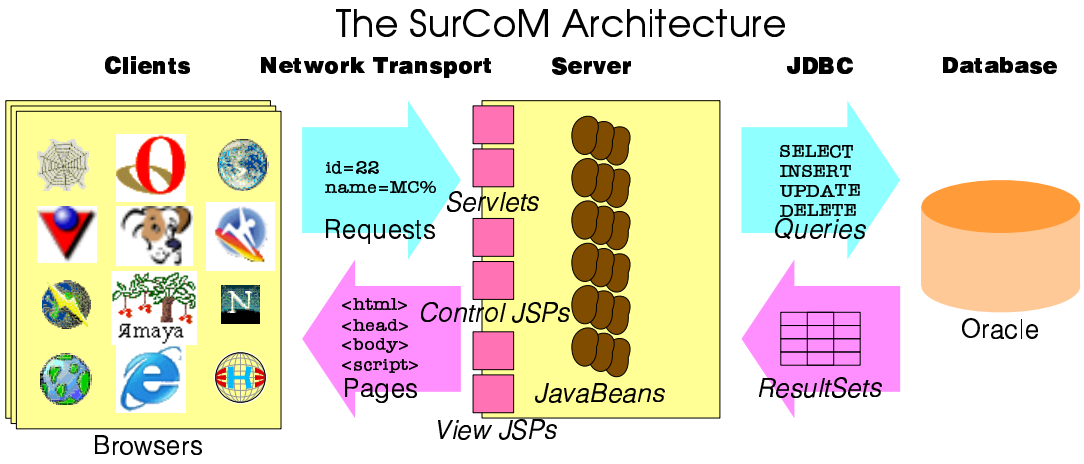


- The STARS architecture, JALife, provides a minimal abstraction layer for JDBC with the goal of allowing result sets to travel from the client to server. Unlike LIMS, no attempt is made to translate business data into object representations, which simplifies the implementation complexity while retaining a level of interface complexity comparable JDBC itself. This is offset by the provision of excellent reusable facilities for data presentation, and a thorough use of object-oriented inheritance throughout.

Apart from the dichotomy of presentation capabilities, these two architectures are comparable is terms of abstraction level. In both cases, very little effort is put into hiding the database implementation behind a facade of any kind. Both architectures would benefit from such a facade, allowing database concepts like cursors to take a back seat to the important issues of business logic. Again, if we ignore presentation capabilities, LIMS' architecture is clearly the preferred candidate for further evolution, being much less tied to JDBC and the underlying database, and taking an object-oriented approach, in what is an object-oriented language after all. (In an ideal world we would like to steal the presentation capabilities from JALife as well, but these are tightly coupled and there are some licensing issues.)

The SurCoM architecture is such an evolution. Much more object-oriented than either LIMS or STARS, SurCoM's data is abstracted into *JavaBeans*[5] which are almost EJBs with Bean Managed Persistence (BMP). Beans begin their lifecycle in their *Handler* (analogous to the EJB home), arising through queries made through their *Manager*. Traversing bean relationships in SurCoM is as simple as invoking find methods (analogous to EJB relationship getters) and setting foreign keys is as simple as invoking describe methods (analogous to EJB relationship setters). The cost is, without the automation of LIMS, all queries must be written by hand, and all beans are populated without the assistance of reflection- a messy implementation to go with the clean interface.

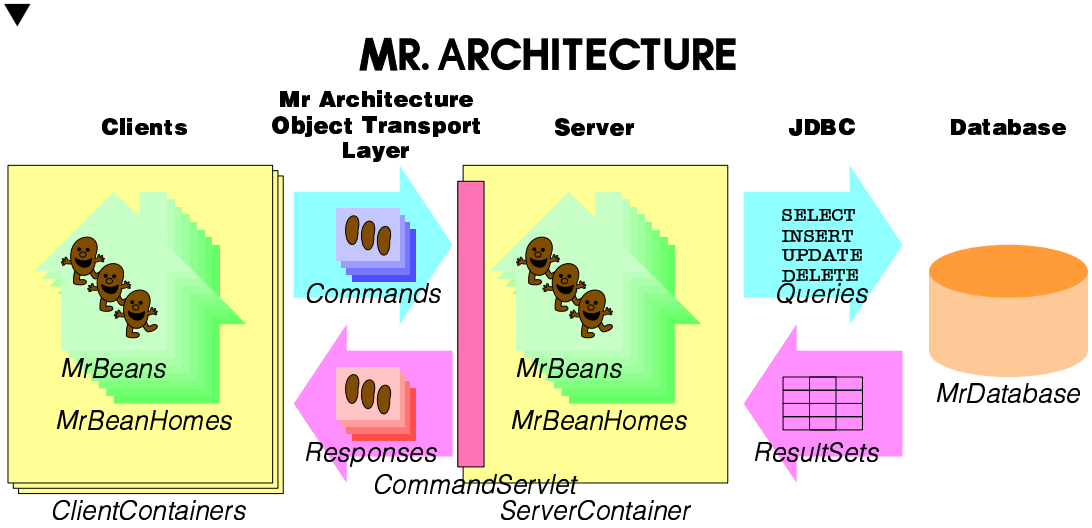
▼



SurCoM also uses *RowObject*, but to no effect, because collections of beans do not have to be serialized- all activity occurs on the server through the action of Java Server Pages (JSPs) and a handful of servlets.

[5] *JavaBeans* were an early addition to the Java 1.1 Development Kit, existing long before Java 2 separated Java into Micro Edition (J2ME), Standard Edition (J2SE) and Enterprise Edition (J2EE). *JavaBeans* are heterogeneous Java objects which can be interrogated dynamically at run time and made persistent in binary files. *JavaBeans* are mainly used to provide user interface components, as EJBs are now used in enterprise frameworks.

In terms of code, the next step up the evolutionary ladder, Mr Architecture, shares only the JDBC abstraction paraphenalia underneath *RowObject* in common with SurCoM, albeit moved into a new package. However, its high concept owes much to pitfalls of SurCoM. Mr Architecture gets much closer to EJB, adopting most of the same structures and naming conventions as needed to support entity beans with Container Managed Persistence (CMP). CMP was chosen to eliminate the maintenance problem facing SurCoM's developers- in Mr Architecture, all code to query the database, unless taking advantage of advanced relational facilities like table joins, is automatically generated. The time saving of this advance alone is worth the price of admission, but Mr Architecture goes much further, providing a seamless object-oriented facade to the relational database.



But even Mr Architecture lacks the elusive presentation layer. This should be set straight in the next evolution, an incremental extension called Ms Architecture, with the power not only to increase the simplicity of future Mr Architecture developments, but also in support of non-Mr applications like SurCoM. Also, to bridge the gap between Mr Architecture and full EJB, Dr Architecture is proposed as a more complete replacement for Mr Architecture. Among its more notable features, Dr Architecture promises to hide transaction management better, while still offering the advanced and highly-efficient bean transport facilities of its predecessor at the implementation level.

What Should the New Architectures Be?

Looking at the implementation of all DPIWE's web-based business applications, there seems to be a tendancy to favour proprietary or in-house solutions for development. Such decisions should not be taken lightly, or at least should be well justified. One justification for deviation on early projects may have been that the technologies required just did not exist. However, today, there is an increasing wealth of both standards and standards-compliant implementations which DPIWE should look to adopt.

In light of this, the most important step for future development in DPIWE to take involves the adoption of most, if not all, elements of the current J2EE standard. The mistake would be, as has occurred and continues to occur in other organisations with similar technologies like Microsoft's *.Net*, to force the adoption from the top-down in a way which would sweep away everything which has gone before virtually overnight. The most important consideration in maintaining a high quality information infrastructure is not to adopt new and emerging standards immediately, but to evolve into them. While a new technology may be inserted almost at the flick of a switch, the people which must understand and support those technologies cannot come out of thin air.

Dr Architecture is a proposed stepping stone into at least one other part of J2EE besides servlets and JSPs- that is Enterprise Java Beans (EJB). Mr Architecture has taken the first step from wholly in-house or proprietary solutions such as those found in STARS, LIMS and SurCoM by attempting to adopt some, but not all, of the EJB 2.0 specification. Dr Architecture must remove the dependencies which exist on a proprietary EJB container and database implementation, or at least remove enough of them to make the final step out of proprietary and in-house architectures as painless as possible.

We can also observe that all the current DPIWE web application architectures since the LIST are three tier:

- Database layer
- Business abstraction layer (LIMS, SurCoM, Tasfol/Vistas) or presentation abstraction layer (STARS)
- Presentation layer

It is clear that the missing element from all but STARS is a presentation abstraction layer. Even STARS could benefit from an enhanced presentation abstraction layer, as it relies on heavyweight clients. STARS would also benefit from a better defined business abstraction layer.

The Ms Architecture Extension is a proposed solution to the presentation abstraction layer question. It is not an entirely new architecture, nor would it be bound to Mr Architecture applications (though it would likely share some code from the *au.gov.tas.dpiwe.mr* package.) It could be used in any application which uses object representations of data, from LIMS to SurCoM to Tasfol/Vistas to future applications based on Dr Architecture, J2EE and beyond.

Improving the Development Methodology- Using Tools Better

Mr Architecture was not just a web application support architecture, and this is very important to note. Mr Architecture is also a methodology for transforming business application concept to business application implementation. It leverages an object model developed in a modelling tool like *Rational Rose*, a tool which DPIWE had already licensed, to rapidly deploy and synchronize both code and database components with minimal translation effort.

This automatic synchronization of electronic model and code is an emerging trend in the development of all kinds of software. It is now supported within many IDEs- Rational just happened to be the pioneer in the area. The savings in development cost and increased level of maintainability of resulting applications by future developers cannot be overstated- the potential for greater efficiency is simply enormous.

LIMS was the first attempt by DPIWE to use such integrated modelling. The *GeoCASE* tool provides some support for generating code templates, but it lacks the ability to synchronize once the code has been developed further. This ends up being a serious impediment to development. The idiosyncrasies of the *GeoCASE* tool and the limited expertise of the current set of developers in using the tool also make it difficult to gain advantage from the electronic model. Latter stages in SurCoM's development were impaired by the combination of reliance on *GeoCASE* and new developers not understanding the tool or relationship between model, database and code.

This highlights the need for integration between the coding environment used by developers, and the design environment used by analysts (who, in DPIWE, are often the same people as those that play the developer role.) Tasfol/Vistas used *Rational Rose* extensively in the early to middle stages, but the perceived difficulty and lack of benefit from using it in the later stages mean that the electronic model was largely abandoned. It is perhaps too early to take a post-mortem of this phenomenon, but it seems that the lack of connection between the toolset developers used every day and the *Rational Rose* tool was the main problem.

It is interesting to note that this time it is not the tool's fault. While some of the developers have a love-hate relationship with *Rational Rose*, most of their criticisms can be dismissed as criticism of how *Rose* fitted into the development process and not of the tool itself.

- One, or at most two machines, are generally available to a project team to run *Rational Rose*. DPIWE has node-locked licenses for reasons of licensing cost, and this severely limits the ability of members of the development team to readily switch to using the appropriate tool for the role they are playing- developer or analyst- which, in DPIWE, can change from hour to hour.
- The machines used to run *Rose* were severely underpowered, running inferior operating systems and having limited processing and memory capacity. For better or worse, *Rose* is resource hungry, and such restrictions prevent *Rose* from performing at a reasonable speed.
- The model was never separated into **petals**, as advocated by the *Rose* consultant early in development, in order to facilitate concurrent development of the model by multiple developers. Part of the reason for this may have been the resource bottleneck of only being able to use one or two machines away from the developers normal workstation- what point, then, is there for facilitating concurrent development?

- The model was never version controlled, and it was not maintained in synchrony with the developed code. One reason for the latter may have been a distrust of the tool by the developers. This distrust was most likely unwarranted, and just an excuse.

DPIWE has a third node-locked *Rose* license, currently utilized on the methodology architect's machine. But even if another dedicated *Rose* machine were established, and there was a small improvement in *Rose*'s utility within the organisation, it may severely impair the architects ability to assist other developers in improving the *Rose*-assisted process. Additionally, the demands on *Rose* are increasing due to a number of small projects now adopting Mr Architecture. To continue to use a modelling tool under such a restrictive licensing arrangement is untenable. If DPIWE cannot afford at least three floating licenses for *Rose*, then it should look to adopt similar tools available from other vendors. Oracle *JDeveloper* claims to have an integrated modelling capacity (in the areas of UML for which Rose is currently used in DPIWE), for example, and this possibility and others should be examined.

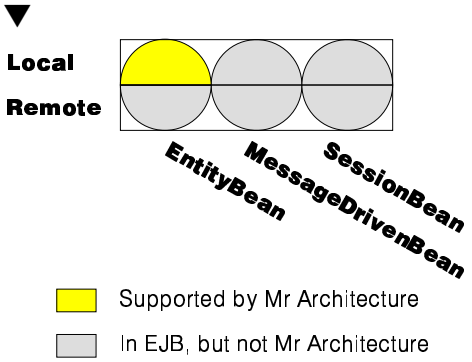
Any modelling solution should be free to use by any developer on their own workstation at any time with minimal contention, otherwise it will not be wholly embraced. It should integrate with their development environment- *Rose* is ideal in this respect because it does not mandate a particular environment and the latest version can interoperate with most of the more popular ones.

This leads to another important point- DPIWE allows developers to choose both the operating system and development environment software they use for development. This greatly increases the productivity of the individual developers, because they are able to use systems which they perceive as most effective in facilitating their daily tasks. However, *Rose* is a mandated tool, and additionally the hardware and operating system is mandated. A person with preference for another operating system faces the double impediment of using a development tool, *Rose*, with which they may not be comfortable, and an unfamilar operating system *as well*. While forcing a developer within a project to standardize on a particular modelling tool may be reasonable, forcing use of a particular operating system or hardware is not. *Rose* has versions for the some of the other preferred operating systems of DPIWE developers, and it may be worthwhile licensing these, or it may be better to choose a different cross-platform modelling tool.

Migrating from *Rose* to another modelling tool would have a slight cost for Mr Architecture- the scripts currently used to transform an object model into Mr Beans would have to be adapted to any new tool. These scripts are not a requirement of Mr Architecture, however, so migration should not be dismissed on this basis alone.

The Trouble with Mr Bean

Mr Architecture is not a bad first attempt at implementing a subset of EJB functionality. Unfortunately, it subsets just a little to much to allow applications built using Mr Beans to be immediately migrated to a J2EE EJB container. If Mr Architecture completely implemented even EJBs with CMP and local interfaces, the task would be much easier. But it does not.



Some of the missing elements are quite deliberately missing- deployment descriptors, for example- in order to ease the burden for developers. But other elements are missing for the sake of making the implementation simpler- Java naming and the semantics of EJB transactions, to name just two.

Clearly both those missing elements which are deliberately missing and those missing just to make implementation easier must be installed into future versions of Mr Architecture if DPIWE is to evolve into using other J2EE implementations. The issue is not so much about extending Mr Architecture, but making applications written in the absence of J2EE features compliant enough to run without Mr Architecture or its future incarnations. To this end, all the Mr Beans currently written and written in future must be translatable into standard entity beans.

It is envisaged that translation will be possible using an automated tool which reads the source files for existing Mr Bean components and produces standard entity bean component source files and deployment descriptors for use in containers other than those of Mr Architecture, such as Dr Architecture. The greatest difficulty will be making applications independent of Mr Architecture's transport layer, and conventions arising out of some of the idiosyncrasies arising from the use of that transport layer, such as `invokeOnServer`, as well as in security and transaction management, such as `deepClone` and `deeperClone`. Automated tools may be able to help here as well, but at least some of the task will need human intelligence.

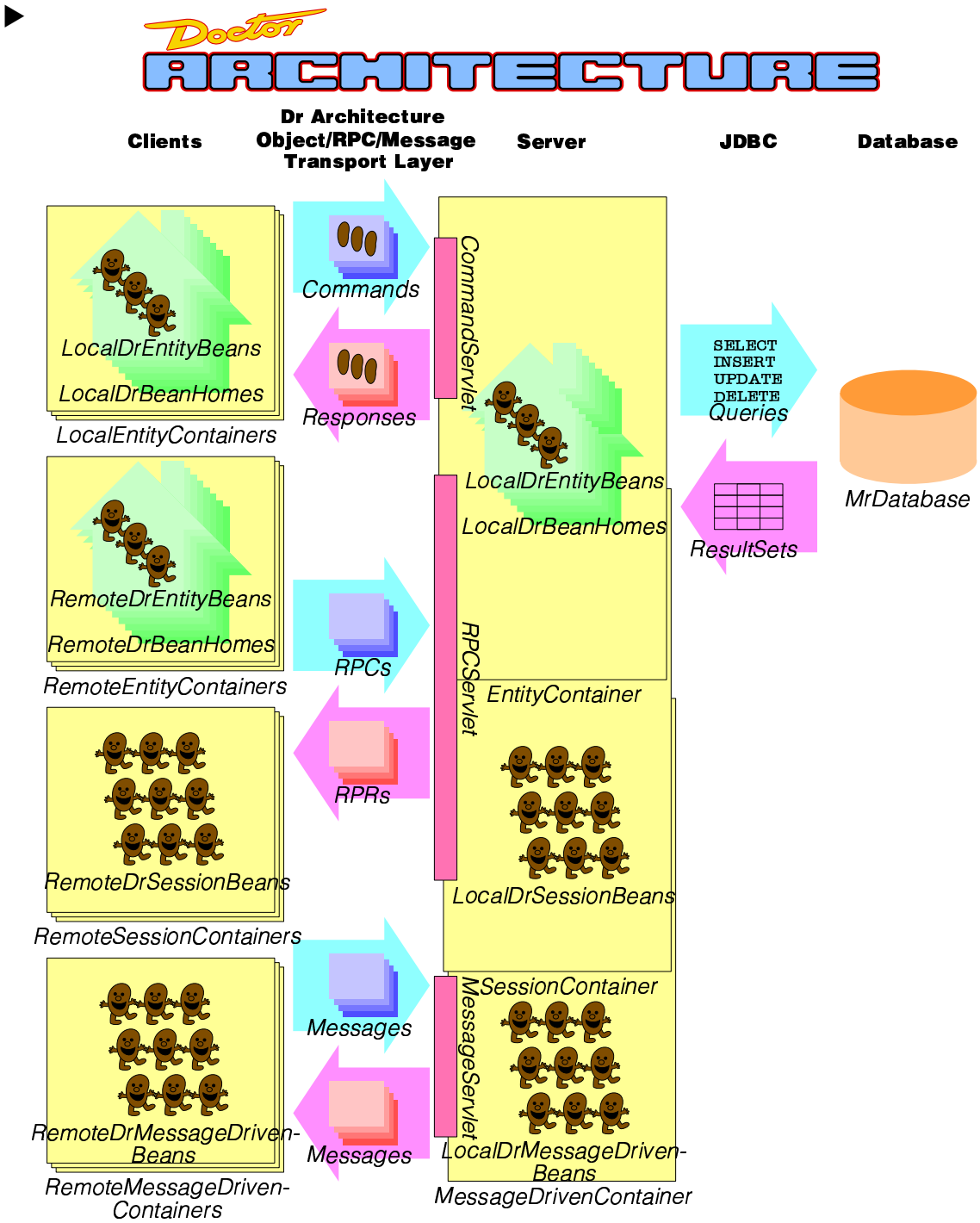
Becoming Standards Compliant- Dr Architecture

One of the impediments to moving into Oracle *9iAS*, IBM *iPlanet*, Sun *ONE* or *JBoss* overnight, besides the new support expertise which would be required of the server team, would be the not insignificant burden on developers to learn a great number of new and emerging technologies while managing to avoid the pitfalls of bending to one implementor's extensions or idiosyncrasies. By implementing an in-house solution, we extend the time allowable for developers to familiarize themselves with J2EE's specification and for the server team to become more adept at providing the support for the new weightier abstraction layer and recommending levels of hardware required to maintain and increase the efficiency of e-business processes.

Dr Architecture is not about reinventing the wheel. The whole point of standards like J2EE is to avoid vendor lock-in and permit a wide choice of implementors. Dr Architecture is about standardizing the development platform of DPIWE to one which can be migrated to a new vendor. Currently this is not possible or would be prohibitively difficult. While we could run our old applications on a different J2EE-compliant servlet engine (like Apache *Tomcat*, as is currently used for newer applications like SurCoM and Tasfol/Vistas) with minimal difficulty, each application would still have to be maintained very differently with respect to their widely divergent architectures. Our aim instead should be to unify the architectures one by one, starting with the least difficult and moving progressively to the more and more divergent, ending with the LIST.

There would be a not insignificant difficulty converting a Mr Architecture application into a Dr Architecture one. The pain for gain required under SurCoM would be proportionately higher, as the architecture it uses is more devolved from J2EE. Converting LIMS would be a significant project. STARS would require an even greater effort. A new J2EE LIST would not salvage a line of code, and would be a functional rewrite requiring time and resources equivalent to the development of the original application. But the benefits to developers would allow them to maintain applications much more easily, the unification of the architecture would greatly reduce the support burden of the server team, and would greatly reduce DPIWE's current shameful dependence on in-house experts familiar with the innumerable idiosyncrasies of the code in each application. The end result for clients would be a direct improvement in the time needed to make business changes both large and small.

Another reason for Dr Architecture is to prove that easing the developers burden does not have to mean an increase in support levels demanded from the server team, nor that unreasonable demands are placed on hardware infrastructure. Dr Architecture would aim to be lightweight and efficient, just like its predecessor, Mr Architecture. Indeed, it could hardly be much more than lightweight with only one or two developers working on the code.



Separation of Concerns

Some important criteria upon which to judge a successful web application are:

- How easily can its style can be changed independent of the content it provides?
    - Style comprises the graphic attributes of data being presented. e.g. colour, size etc.

- How easily can its content can be changed independent of the business logic it implements?
    - Content comprises the semantic attributes of data being presented.
      e.g. this is a person's name, this is a geographic coordinate etc.
- How easily can the business logic be changed independent of content and style?
    - Business logic comprises the dynamic behaviour of data.
      e.g. how to process a payment, how to store a coordinate, etc.

In a lightweight web application, where some mark-up language is presented for the client browser for rendering, and requests are generated from the browser software under control of the mark-up language (and its script elements), the style and content are combined either in the browser (with CSS), in the server (via XSL transformation) or some combination. It is also possible for these to be precombined, but this is not ideal because it means that the answer to the first question is "not very easily." Unfortunately, many of the DPIWE applications which use a lightweight approach (the LIST, LIMS and SurCoM) are programmed this way.

In a heavyweight web application, where software is installed within the client browser to manages the user interface (UI) independently of the browser's facilities, the division of style and content is the responsibility of that software. For example, Tasfol/Vistas delegates style by instantiating styled components, which know how they are to be presented under the current stylistic guidelines, to build up the UI content.

In either case, how well the business logic is separated from content and style is not a matter of technology, but a matter of programming style.

The LIST and LIMS deliver HTML content, but not to any W3C standard format. Most of the content has been written for and tested on a limited subset of browsers, and it is only by virtue of being quite a bit older (and therefore using an older and more widely adopted form of HTML) that the LIST manages to run on most browsers.

In the LIST, pretty much all the style, content and logic is embedded in the generating PL/SQL code. There are, of course, minor miracles you can perform with Perl to suitably mangle the output, but pretty much what you are left with is a completely unmaintainable amalgam of what should be separate concerns. LIMS is pretty much in the same boat, with most style, content and logic embedded in Java servlets, a sprinkling in Java applets, and in (pretty much) god-knows-what-else. One further complication in both these systems is the heavy use of client-side technologies like JavaScript. In the LIST, these extensions work in most browsers, but in LIMS you are stuck with IE.

STARS is an interesting case. There is evidence of some business logic being factored out into controller objects, but the main control flow of the application relies on UI components undertaking to obey business rules within themselves. So it is a halfway house where some business logic can be affected independent of presentation components, but most of the important stuff is embedded in the UI. Not that STARS really deserves to be criticized for this, as Tasfol/Vistas suffers from much the same syndrome, though arguably to a lesser extent- see below.

SurCoM uses XHTML as its primary content delivery mechanism. (It also has the capacity to deliver CSV data, but this is not as pervasive.) Unfortunately, the mark-up is pure HTML, with no separation of concerns- the JSPs are programmed with both the content and presentation logic intertwined, with most (but not all) style contained in a single separate cascading style sheet (CSS). However, at least by using XHTML, there is some possibility that SurCoM could be unravelled using an XSL transformation with just a sprinkle of human intervention. Also, it is probably unfair to say that all logic and content is intertwined- SurCoM uses the model-view-controller pattern to separate most high-level logic concerns from the content. The problem is specifically that the view JSPs have logic in them as well as the controller JSPs, when really this should all be in the controller JSPs or a third thing.

Tasfol/Vistas does a better job of separating content from logic than most of the foregoing applications, but there is still some blurring at the edges. Ideally, all business logic in Tasfol/Vistas would be embedded in the beans, and all presentation concerns would be left to the UI classes. But in reality, the distinction is often not so clear. This is one area, at least, where SurCoM comes out on top- pretty much all the business logic in SurCoM is in the controller JSPs and the library classes (including the beans encapsulating records from the data model), and content is almost the exclusive domain of the remaining view JSPs. One criticism of SurCoM is that it puts too much logic in the controllers and not enough in the beans. Tasfol/Vistas strikes more of a balance here, but sacrifices the (arguably more important) view/controller distinction.

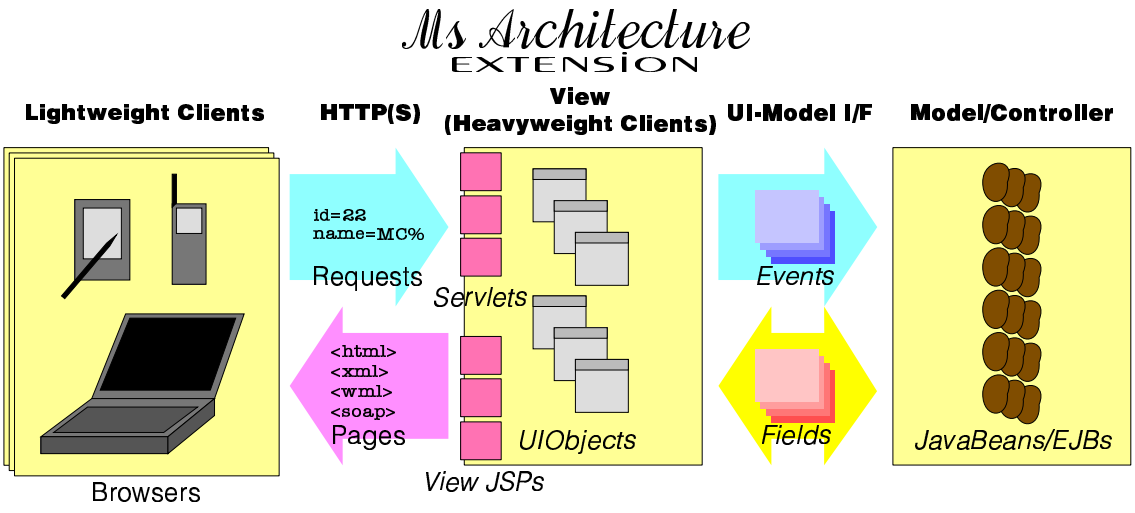
Mr Architecture must wear some of the blame for the Tasfol/Vistas view/controller blur, because it does not support session beans. Session beans are commonly used in EJB applications to encapsulate controller functionality, manipulating groups of beans according to business rules. Without them, a business method that does not belong in any entity bean is very easily placed in the UI class which is most often responsible for initiating the behaviour. That said, it probably wouldn't be too hard to move these methods to session beans as they become available under evolutions of Mr Architecture (like Dr Architecture.)

Separating Content from Logic- The Ms Architecture Extension

SurCoM, and future lightweight web applications, would benefit from a presentation logic abstraction layer that removed all logic from their presentation components (in SurCoM's case, view JSPs.) If such a presentation logic abstraction layer was sufficiently general, it could also be used to generate source code for the UI components needed in heavyweight web applications.

Such a presentation layer is proposed as part of the Ms Architecture Extension. Ms Architecture would not rely on Mr Architecture directly, but would use bean introspection to extract data from object representations of data in whatever form: plain objects, the JavaBean-cum-EJBs using BMP found in SurCoM, the almost-EJBs using CMP found in Mr Architecture, or full-blown EJBs.

▼



Unlike Mr Architecture, and other more complete EJB implementations, which mandate a static deployment step, the Ms Architecture presentation layer will have the flexibility to operate with or without such a step. Therefore, three modes of operation of Ms Architecture are proposed:

- **Fully static mode**, which takes a bean class as input and produces a template (either source code for a UI component, or skeletal mark-up language content) into which bean data can be inserted at run time.
- **Slave mode**, which provides a JSP tag library which can be used to introspect beans at run time and produce dynamic content. This tag library is used in the JSP content forms produced in fully-static mode.
- **Fully dynamic mode**, which takes a bean object available at run time and produces a view object or some mark-up which presents the bean content in a particular form. Code from bean class doer methods will rely on methods available in fully dynamic mode in order to initiate further user interactions.

UIs are additionally presented in a number of modes:

- view only mode (`UI_ViewOnly`),
- view/edit mode (`UI_ViewEdit`), and
- search mode (`UI_Search`).

Each possible UI interaction is then mapped onto a bean which is presented in one of these modes:

- An introductory or help screen would be modelled as a bean displayed in view only mode with one field containing the text required.
- A menu would be modelled as a bean with no fields but having a doer method corresponding to each menu option.
- A search or data entry screen could be modelled directly using an entity bean (like a Mr Bean), or indirectly through a bean specically crafted to combine relevant fields ranging across many entity beans.

As an example of how powerful this approach is, it would be possible to completely decouple the user interface of SurCoM form its business logic layer by recasting it to use Ms Architecture, because SurCoM consists solely of interactions of the types already described (with some minor exceptions to do with file import/ export facilities).

Examples of Fully Static Mode

This mode will be most familiar to Mr Architecture developers, as it resembles the bean deployment step. Indeed, the generation of Java source code from this step will likely utilize primitives already available in Mr Architecture.

In this mode, Ms Architecture's *StaticTranslator* is presented with a compiled bean class, similar to the way an abstract compiled bean class is provided to the Mr Architecture *DeploymentTool*, and then proceeds to generate a static template which can be used at run time to present the data.

One possibility offered by this mode is customization of the generated template by developers. With this added power comes the responsibility of deciding when it is appropriate to modify the generated template. Modification of the template will almost certainly rule out future automated translations from model to UI, for example.

For example, an *OrderForm* bean may be transformed into an *OrderFormUI* class which extends *JPanel*, and takes an *OrderFormObject* and the UI mode as parameters to the constructor. The *OrderFormObject* is the model, and the *OrderFormUI* is the view. Any control actions are defined by doer methods in the bean which are relevant to the UI mode specified. e.g. a **Search** button will be provided on the panel when instantiated in `UI_Search` mode, and clicking on it will activate the `doSearch` method.

If a bean is statically converted to XHTML, WML or XML, then each leaf element will have no content except for a comment "<!-- `Insert field here` -->". If converted to WML or XHTML, then three variants are produced- one for each UI mode.

An example of the static XML produced for the *OrderForm* bean might be:

```
<OrderForm>
        <Customer>
                <!-- As specified by getOrderFormFields() -->
                <code autoMatch><!-- Insert field here --></code>
                <name autoMatch><!-- Insert field here --></name>
        </Customer>
```

```
        <OrderLine repeatable visibleRows="5">
                <Product>
                        <code autoMatch><!-- Insert field here --></code>
                        <description autoMatch><!-- Insert field here --></description>
                        <unitPrice readOnly><!-- Insert field here --></unitPrice>
                </Product>
                <quantity><!-- Insert field here --></quantity>
                <totalPrice derived><!-- Insert field here --></totalPrice>
        </OrderLine>
        <orderTotalExTax derived><!-- Insert field here --></orderTotalExTax>
        <orderTax derived><!-- Insert field here --></orderTax>
        <orderTotalIncTax derived><!-- Insert field here --></orderTotalIncTax>
</OrderForm>
```

A bean may also be statically converted to an XHTML JSP, a WML JSP, or an XML JSP. In this case the bean content is read from the attribute au.gov.tas.dpiwe.ms.currentBean in either the page context, request context, session context or application context (depending on a switch), and JSP tags available in slave mode are generated as content for each leaf element.

An XML JSP for *OrderForm* might look like this:

```
<jsp:root
        xmlns:jsp="http://java.sun.com/JSP/Page">
        xmlns:ms="http://developers.dpiwe.tas.gov.au/Ms/JSP">
        version="1.2">
<jsp:useBean
        id="au.gov.tas.dpiwe.ms.currentBean"
        class="au.gov.tas.dpiwe.ms.example.OrderForm"
        scope="request" />
<OrderForm>
        <jsp:declaration>
                au.gov.tas.dpiwe.ms.example.Customer customer=currentBean.getCustomer()
        </jsp:declaration>
        <Customer>
                <!-- As specified by getOrderFormFields() -->
                <code autoMatch>
                        <ms:format-field>customer.code</ms:format-field>
                </code>
                <name autoMatch>
                        <ms:format-field>customer.name</ms:format-field>
                </name>
        </Customer>
        <jsp:declaration>
                java.util.Collection orderLines=currentBean.getOrderLines()
        </jsp:declaration>
        <ms:for-each
                atLeastOnce
                item="orderLine"
                type="au.gov.tas.dpiwe.ms.example.OrderLine"
                group="orderLines">
        <OrderLine repeatable visibleRows="5">
                <jsp:declaration>
                        au.gov.tas.dpiwe.ms.example.Product product=orderLine.getProduct()
                </jsp:declaration>
                <Product>
                        <code autoMatch>
                                <ms:format-field>product.code</ms:format-field>
                        </code>
```

```
                    <description autoMatch>
                        <ms:format-field>product.description</ms:format-field>
                    </description>
                    <unitPrice readOnly>
                        <ms:format-field>product.unitPrice</ms:format-field>
                    </unitPrice>
                    <taxable>
                        <ms:format-field boolean>product.taxable</ms:format-field>
                    </taxable>
                </Product>
                <quantity>
                    <ms:format-field>orderLine.quantity</ms:format-field>
                </quantity>
                <totalPrice derived>
                    <ms:format-field>
                        product.unitPrice*orderLine.quantity
                    </ms:format-field>
                </totalPrice>
            </OrderLine>
            </ms:for-each>
            <orderTotalExTax derived>
                <ms:format-field>Sum(orderLines.totalPrice)</ms:calculate-field>
            </orderTotalExTax>
            <orderTax derived>
                <ms:format-field>
                    au.gov.tas.dpiwe.util.Taxes.taxPayable(orderLines{taxable=true}.totalPrice)
                </ms:format-field>
            </orderTax>
            <orderTotalIncTax derived>
                <ms:format-field>
                    currentBean.orderTotalExTax+currentBean.orderTax
                </ms:format-field>
            </orderTotalIncTax>
</OrderForm>
</jsp:root>
```

Tags Available in Slave Mode

Static mark-up language templates are only useful if they can be converted to concrete mark-up at run time. There are many solutions to this problem, most involving SGML or XML transcoders. Ms Architecture will provide one possible solution- JSP templates produced in fully static mode will be converted through the action of a custom tag library.

To this end, the following tags are provided by slave mode, which are also invokable from manually created JSPs:

- Formats a numeric or string field. The `format` parameter is ignored in the case of a string field.

```
<ms:format-field format="Java decimal format">
        orphan expression in Ms Architecture format
</ms:format-field>
```

- Formats the first non-null numeric or string field from a list of possibilities. The `format` parameter is ignored in the case of a string field.

```
<ms:choose-field format="Java decimal format">
        collection expression in Ms Architecture format
</ms:choose-field>
```

- Displays one or more possible non-null values for a numeric or string field. The format is ignored in the case of a string field.

```
<ms:list-fields
        format="Java decimal format"
        layout="one of table, definition, ordered or unordered
                list, paragraphs, lines or CSV"
        style0="style name to apply to first possibility"
        style1="style name to apply to second possibility"
        ...>
        collection expression in Ms Architecture format
</ms:list-fields>
```

- Displays the content multiple times, once for each member of an iterator, array, map range or collection, initializing a scripting variable to refer to the current member. When atLeastOnce is present, an empty member is generated if the iterator, array, map or collection is empty.

```
<ms:for-each
        [atLeastOnce]
        item="scripting variable to refer to current member"
        type="scripting language type of members"
        group="scripting variable corresponding to group">
        arbitrary content, including other ms tags
</ms:for-each>
```

- Displays content only if a scripting variable references a value.

```
<ms:if-there-exists
        item="scripting variable"
        suchThat="logic expression in Ms Architecture format">
        arbitrary content, including other ms tags
</ms:if-there-exists>
```

For an example of usage, refer to the XML JSP for *OrderForm* in the previous section.

Methods Supporting Fully Dynamic Mode

An application constructed around Ms Architecture can be dependent on the presentation device or independent of it. Clearly, it is not wise to be wholly device-dependent, but sometimes it can be advantageous to use the full capabilities of an available device. Where a device-dependent interface is constructed, a device-independent alternative with possibly restricted functionality should also be provided. Object-oriented inheritance is perfect for creating such an arrangement without the need for duplicate UI code.

Applications based on Ms Architecture will call methods within Ms Architecture at run time to generate presentation objects or initiate user interface interactions. To provide an insight into the baseline level of user interface allowed for by Ms Architecture, some examples of the proposed methods are:

`JPanel SwingUIBuilder().buildUI(JPanel panel, Object bean, int uiMode)`	Build a user interface in `panel` and send events to doer methods in `bean`.
`void MarkupUIGenerator(String outputType[, String statusMessage]) .presentUI(OutputStream out, Object bean, int uiMode)`	Presents a user interface to a remote lightweight client which is using the `outputType` markup language. User interactions arrive back at the *UIServlet* and result in events being sent to doer methods in `bean`.
`Collection PresentationManager() .setMenuOptions(Collection menuOptions)`	Sets the collection of *MenuOption*s which are presented to the user in the current manner for this user's session.
`Collection PresentationManager().getMenuOptions()`	Returns a collection of *MenuOption*s which may be mutated- these are presented to the user in the current manner for this user's session.
`void PresentationManager() .setStatusMessage(String statusMessage)`	Presents status message in current manner for this user's session.
`void PresentationManager().showUI(Object bean, int uiMode)`	Presents user interface in current manner for this user's session.

Conclusion

This paper has focused on three key weaknesses in DPIWE web application development, and has proposed two exercises which will hopefully address two of these:

- Dr Architecture, to address the problem of migration to standard J2EE application server arrangements.
- The Ms Architecture Extension, to address the issue of separation of presentation and business logic concerns within applications.

The third weakness is in the use of modelling tool Rational Rose and its weak integration into developer workflow. We do not blame the tool, merely the way it is licensed and deployed within DPIWE. Some solutions have been suggested, but these will have to be considered further before a decision is taken.

This paper does not seek to imply that these are the only weaknesses in DPIWE web application development, nor that they are the most important. They are, however, certainly among the most important. One thing that can be said about all three is that there is a way forward.

If Dr Architecture is permitted to be the next evolution of the DPIWE web application infrastructure, and the Ms Architecture Extension is applied pervasively to address separation of concern issues, then the already successful range of CIT-developed applications can only grow and improve. At the same time, reliance on proprietary technologies and in-house experts to know and understand poorly structured or documented code should also be greatly reduced.