White Paper

Organizational Structure and Repository Governance

WP0198 | September 2015

Peter Harrad

Peter has worked with modeling standards and techniques throughout his 20 years in IT, in a career that has covered software development, solutions architecture and international consulting.

Peter's particular areas of interest are opportunities arising from interdisciplinary touchpoints, how to balance practicality and rigor when modeling, and the importance of viewpoints in addressing different stakeholder perspectives.

Having a clear view of the current state architecture provides an extremely valuable resource for planning new implementations and responding to unforeseen circumstances – whether it's a subsidiary that is being sold and needs to disambiguate their systems, a company scrambling to fulfill an impulsive promise made by the CEO on television, or a government department working to deal with some government crisis.

However, the problem with keeping a model of the current state is that the reality is constantly being updated, whether the model is or not. New implementations are rolled out from various groups on an ongoing basis, which means that the organization has to invest effort in maintaining the current state. Unless this is done in a formal way, with consideration of how the architecture function will govern the process of maintaining the current state, updates will not be made. More than once I've encountered an organization with a beautiful, detailed model… that just happens to be inaccurate because it's not been kept in sync with the reality. In which case, why bother in the first place?

In this paper, I will examine the challenges that exist, consider the two main operating models and discuss what this implies for the features you should look for when selecting the main architecture repository and associated tooling.

Challenges with Maintaining Current State

The analogy of rebuilding a 747 in flight is used enough that it has almost become a cliché, but like most clichés, this is because it is so apt. Projects are updating the current state on a regular basis. Different groups will quite likely be making updates on different schedules. At one organization where I worked, the infrastructure group released into production project by project, as they were ready. The applications division released once a month, every month. Except the directory services part of the infrastructure division released with the applications division releases, not on the model of the rest of their colleagues.

What this means is that without completely reworking the operating practices of parts of the IT function, the mechanism for updating the current state needs to take account of multiple, different update schedules.

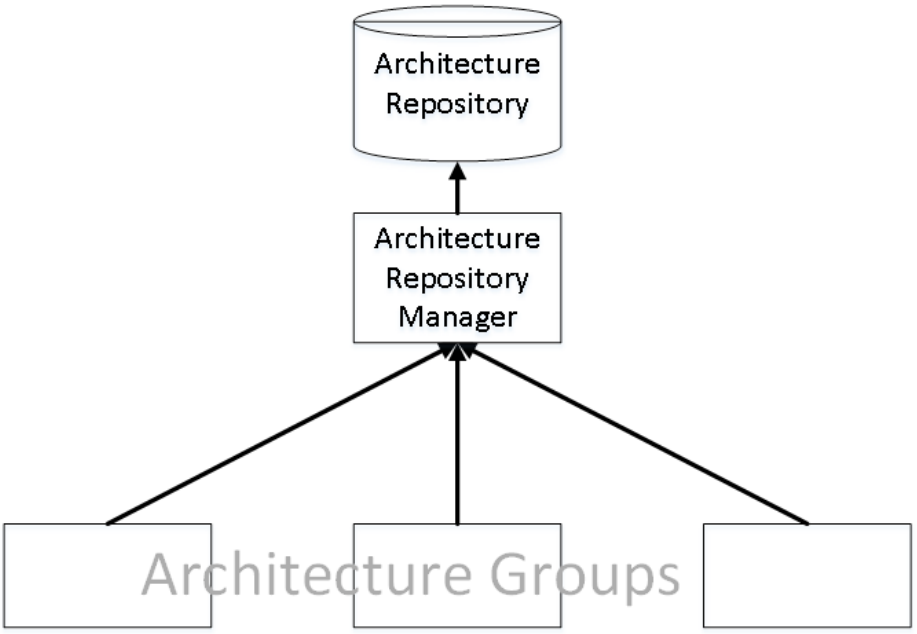
Now, possibly the key challenge that an organization faces in maintaining a view of its current state architecture is that the architects who are making changes are busy individuals. When you're under a time pressure, you look for ways to save time, and when a solutions architect is faced with 6 projects, all requiring attention, it can be all too tempting to put off updating a model 'until tomorrow'. But tomorrow never comes. Even if the architect buys into the need to provide an update, does the project manager? What about the project sponsor? It's all too possible for updates to slip.

This means that there is a need to audit the projects that have gone live and check that the current state reflects any changes that have been made. Who performs this audit function will vary according to which of the models below the organization has adopted. One technique that I've seen adopted is to leverage the project tracking tool that the organization uses, to include a way to record that the project has taken the necessary actions required of them to support the current state maintenance.

The next issue with maintaining the current state is one of notifications. As noted above, a significant reason for maintaining a current state is to enable projects to plan the organization's implementations more effectively. This means that as the current state is updated, assumptions that projects have made about the environment that they are deploying into may no longer be valid. So ideally, governance of the current state model should include a mechanism for projects to be made aware of what change have been made, to enable them to perform impact analysis.

Centralized Current State Maintenance

The first way that organizations choose to maintain the current state is to have a centralized individual or team that is responsible for maintaining the current state. This is often found where there is a formal enterprise architecture team, and maintaining the current state forms part of their responsibilities. In this model, projects provide their architectures to the central group, who update the current state model as each project goes live.



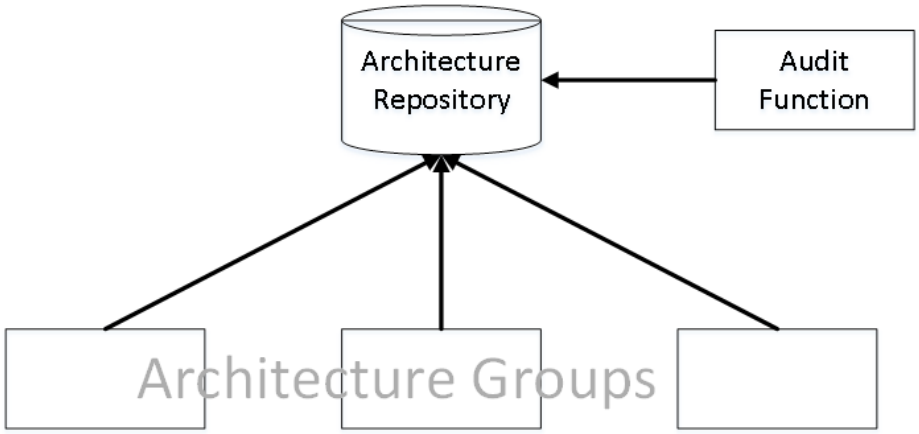
Centralized Maintenance

This approach has the benefit that the people making the updates will see it as a core aspect of their job (and if they are a true EA team, charged with transformation as well, it can provide a useful tool for such activities). The risk is that the central group is in danger of being seen as yet another administrative burden – the 'high priest' syndrome. To counter this, the central group should look for ways to support the projects – for example, by helping with specific analyses. One example I'm aware of is a US federal body; when the US government had a shutdown in 2013, the chief architect (and manager of the current state) was able to assist various groups in prioritizing which services to shut down, based on his understanding of which user communities used which services.

Distributed Current State Maintenance

Where things get more interesting is with the distributed approach. Essentially, each architect is responsible for updating the current state as their individual project goes live. This tends to exist where there is no central function that could be assigned the role of performing updates – hence the responsibility for maintaining the repository becomes a collective one.

At first glance, on paper this model can seem more attractive to management, as it appears to distribute the extra workload fairly, adding a small amount of overhead to each participant. However, it turns out to require a level of extra effort – the need to audit and train staff.



Distributed Maintenance

The drawback with this approach, is that as mentioned above, if each architect is responsible for making their own updates, there is the risk that updates will not be made - for the reasons discussed earlier (time pressure and higher priority tasks crowding out what is seen as an administrative task). This means that whereas in the central model, there is just a need to check that updates have been provided for each project, something that can be done simply by checking email or file systems, here *someone* needs to check that the updates have been made to the current state. Whether this should be a role that is permanently assigned to a group or individual, or a rotating job within each division, will depend on the culture of the organization and managerial preferences.

The second consideration that appears in this approach is that merging changes into an overall model can be complex. Architects do tend to be an intelligent set of people, but merging changes in like this is yet another hat for busy people to wear. The additional overhead can be helped by providing a simple process map and set of guidelines for architects to follow when making their updates.

Tool considerations

The past discussions have highlighted a variety of activities that different roles within the organization need to undertake. Given that this topic is centered on managing the architecture models, it is worth considering how the architecture repository and modeling tool functionalities can make the task easier. Actions such as updating the current state, performing an audit and enforcing governance are all areas that can benefit from specific tool functionality.

The first consideration that exists is that if the current state (describing the current reality) and a project architecture (describing the planned

reality) are to be managed within the same repository, the repository needs to be able to keep multiple separate versions of the same item. This allows projects to plan changes without actually editing the current state model. For example, a given application might have a new interface added by a project; until the interface is rolled out, the current state should not reflect. Several of the repository-based modeling tools that exist offer some mechanism to define different partitions with separate copies of objects. For example, iServer has the libraries facility.

Keeping planned and current states separate is useful, but as discussed earlier, the planned state changes will need to be applied to the current state when the project becomes live. When this happens, the task is greatly simplified if the tool has the ability to compare two models and provide a list of differences – a gap analysis, in other words. TOGAF does provide a template technique for gap analysis in chapter 27, but the matrix format it uses is not ideal for readability: I have found that a list format, as used by a couple of tools, works better from a usability perspective.

The counterpart to the compare functionality is the ability to merge changes from one model into another. This should never be a dumb, purely automatic process; instead the architect needs the ability to veto a given change being made (e.g. if a software level is upgraded by a project, but it's already been taken to an even more recent version, we would not want to the merge functionality downgrading it). In an ideal world, there would be a list of changes and the ability to confirm or deny each one.

Notifications and conflict detection are another area where a tool can greatly ease the burden. Specifically, the ability for architects to get notified when certain items are updated in the current state; or even notified when other planned architectures are using the same entities as their project. This is, perhaps surprisingly, not a common feature currently found in tools at this time; in such a case, an alternative approach is to have access to a report that allows you to see what entities have been updated in the current state since a given date – or what other projects are planning changes to the same entities.

Depending on which model, centralized or distributed, exists, access permissions may be extremely useful to ensure that only those who are authorized to update a model, can update that model. This can also be a requirement for organizations engaging in government work.

Last of all, and more important than access permissions is the tool's support for audit trails. A good tool will provide a record of each change made to an entity, with a timestamp and the identity of the person making the change.

Summary

The mechanisms for updating the current state model may need to cope with different cadences of updates and ideally should enable notification of changes to interested parties.

Organizations need to have an audit function to ensure that updates are being provided (in the centralized model) or made (in the decentralized model). This could be a single individual or group, or a rotating position – the culture of the organization will affect this.

If centralized management is adopted, the central function should offer ways to assist the functions, e.g. by using their specialized knowledge to assist reporting.

Adjusting the organization's project tracking tool to record that the updates for the project have been provided (or made) is one way to ease the audit burden.

Where the project review process is documented, a small change to the process that includes having the project architect either making or providing their update cements the need for this step.

Architecture repositories can assist the process of maintaining the current state by providing functionality such as: partitioning, compare and merge capabilities, notifications, audit trails and access control.

References

TOGAF chapter 27 – Gap Analysis

TOGAF chapter 41 – Architecture Repository

TOGAF chapter 50 – Architecture Governance

"A Federated Approach to Enterprise Architecture Model Maintenance" - Ronny Fischer, Stephan Aier, Robert Winter

Orbus Software UK
London

Orbus Software US
New York

Orbus Software AUS
Sydney

Orbus Software RSA
Johannesburg

enquiries@orbussoftware.com | **www.orbussoftware.com**

Seattle Software Ltd. Victoria House, 50-58 Victoria Road, Farnborough, Hampshire, GU14 7PG. T/A Orbus Software. Registered in England and Wales 5196435