Application Landscape Views and SOA

Ideas on dealing with challenges posed by the proliferation of services/API connections

Table of Contents

Table of Figures

Introduction

As part of understanding how an organization works, enterprise architects describe and visualize its "application landscape". These descriptions and visualizations are meant to cover the most important aspects of enterprise applications. The content reflects architects' concerns and their perception of importance; naturally, not everything regarding enterprise applications can or needs to be covered.

Looking at the matter from the perspective of application integration, I have noticed that application landscape views seldom show the way applications are interconnected by web services, let alone the interaction of services across organizations' boundaries. The common way of examining application landscapes hence insufficiently reflects the importance of the proliferation of service/API interactions.

The above statement is based on my own experience, discussions with colleagues, and on publicly available online sources. While this may not count as evidence, it is safe to assume that many larger organizations have limited insight into the service interactions of their applications.

The question thus arises: What damage may this lack of insight cause? It is hard to give a precise answer, even after thorough research. However, it is reasonable to suppose that such a lack may lead to negative consequences and associated risks.

This article proposes using an architecture view of ArchiMate®[1] to create insights that address the implications of a large number of web service interactions. The purpose of these insights goes beyond architecture modeling and communicating the results of architects' work. The same insights are needed by different stakeholders on different levels of an organization such as IT operations, especially when dealing with an "as-is" application landscape. In other words, the article implicitly proposes extending the use of the ArchiMate language and its concepts beyond architecture processes.

"Application Landscape" and Service Orientation

The term "application landscape" is in frequent use, especially when talking about managing the application portfolio of an organization. What does this term mean? And why do we use the word "landscape" metaphorically – a question that can be applied to other contexts, for example when talking about a "political landscape"?

After looking through some publicly available online sources, I was able to find a definition that reflects the common usage:

> *"the entirety of all business applications and their relationships in an organization"*

[Bu01]

The necessity to look at the "entirety of all business applications" is strongly associated with a reliance on SOA-based solutions. While the acronym SOA isn't much used lately, this does not mean that service-oriented architecture has disappeared overnight. On the contrary, it indicates that the SOA hype is over, and that this architecture has matured and entered the mainstream.[2] Thanks to SOA (backed by solid web services technologies), IT solutions have gained in strength and flexibility. Advancements such as enabling access to shared data across an organization, providing online services to partner organizations, or facilitating multi-channeling by exposing back-office services to the front-office are all possible due to service orientation. In short, SOA has allowed business functionality to be supported by multiple software components that interact without human interposition.

This creates dependencies. Work can only be carried out if all applications and their interconnections in the process chain are functioning correctly. Applications use services provided by other applications, and

[1] ArchiMate is a registered trademark of The Open Group.
[2] See, e.g., Gartner (https://www.gartner.com/doc/1824514/magic-quadrant-soa-governance-technologies)

cannot fully perform their tasks unless the services they use are also functioning. This functional dependency has become an essential aspect of the way in which organizations work.

Modern business processes therefore depend not only on the functioning of each individual application involved but also on interconnections between applications and, moreover, on functional coherence across multiple applications. Hence, in respect of service-based interactions, organizations need to control their applications as an integrated whole and view them as an "application landscape".

The Growing Importance of Seeing the Interconnected Whole

What is certain is that larger organizations use many web services[3] and that this usage is growing. Moreover, certain IT trends are bringing new challenges to this domain, namely:

- More web services across organizational boundaries
- Service exposures to mobile devices
- "Open Data"
- Microservices
- Mixed application deployments (on premise, PaaS, IaaS, SaaS, etc.)

One aspect that these trends have in common is their contribution to the complexity of application landscapes. Not only are applications functionally entangled with one another, but the application landscape is also diversifying. On the one hand, technology simplifies the way we establish and manage individual service interactions;[4] on the other, the overall complexity increases.

This should be a cause for concern, and is best illustrated by way of some examples. We should ask ourselves whether we know:

- How many service connections exist, and can they all be audited (performance, reliability, security, etc.)?
- What are the consequences of an application's failure for the rest of the application landscape, and which business processes are being affected? What are the dependency chains?
- What is the dependency of my organization on services provided by other organizations and vice versa? Which of my organization's services are exposed to consuming applications over which I have no control?
- Is the use of services by mobile devices visible in architecture views, and are these adequately managed?
- Which mechanisms and products (in different hosting environments) are being used to secure services and restrict access to them? Which services are being accessed without mediation by a security gateway and what are the security risks involved?
- In terms of service interactions, is there a significant gap between the real-life application landscape and the models maintained by enterprise architects, which are believed to reflect the "as is" situation? Is the organization entangled in "spaghetti SOA" without knowing it?

This list emphasizes the importance of understanding service interactions and highlights the fact that their relevance extends beyond the "going concern". Even if we are engaged in a major transformation, the existing situation must provide the foundation. As the saying goes: "You can't get where you want to be if you don't know where you are".

---

[3] The term "web service" in this context denotes all web APIs, disregarding protocols and standards (XML/JSON, OpenAPI/WSDL etc.). The fast increase in popularity of JSON, OpenAPI (Swagger) and RESTful, all of which are significantly related to service interactions, is beyond the scope of this article. Discussing these phenomena would distract from the subject at hand.

[4] "Interaction" in this context refers to the plain English use of the word. It is by no means related to the ArchiMate element "Application Interaction".

In short, we should be aware that the proliferation of web service interactions is one of the strongest forces shaping application landscapes. Consequently, an accurate model of service-based application-to-application connections is a vital component of application landscape insight.

Application Landscape And ArchiMate

Whatever perspective we choose for looking at an application landscape, we need to see its structure. Structure means architecture, and most take an architectural approach, namely by using viewpoints and views[5].

Online sources reveal that architects have come up with a variety of views, as they require different insights to deal with a range of possible concerns. Most commonly, these views show some clustering of applications based on their associations with business processes, business units and/or information domains.

When it comes to visualizations, these views are expressed using different notations. It goes without saying that adherence to a standard is an advantage, not least for its predefined common vocabulary. And when it comes to standards for documenting architecture, the natural first choice is ArchiMate, a visual language for architecture modeling and description.[6] An Open Group standard, ArchiMate is gaining traction, especially in Europe. It provides a list of standardized viewpoints [Og01], one of which is the viewpoint called "Application Co-operation", which has been conceived for our purposes here (the documentation states: "This viewpoint is typically used to create an overview of the application landscape of an organization.").[7]

The following diagram shows the elements that may be used in views governed by this viewpoint:

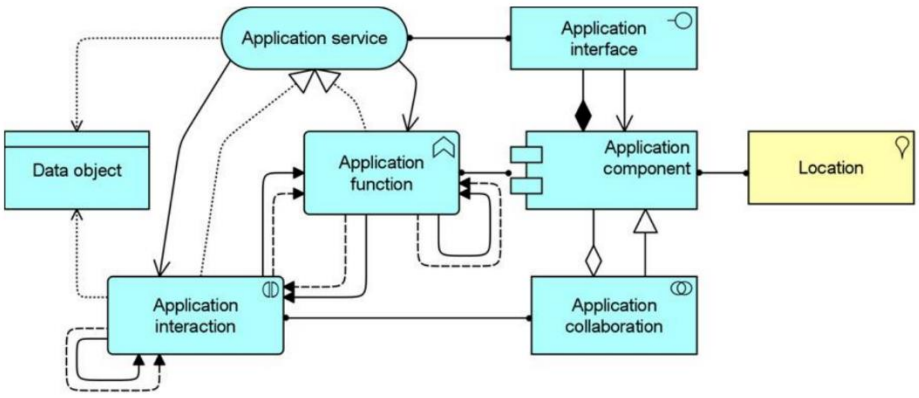

Figure 1 Concepts and relationships in the Application Co-operation Viewpoint[8] (ArchiMate 2.1)

[5] Also standardized in ISO/IEC 42010.

[6] Other architecture standards and even UML would likely suffice to describe the dependencies caused by service interactions. However, this discussion is beyond the scope of this article.

[7] Viewpoints are extensively elaborated in "Enterprise Architecture at Work" [Lk02], chapter 8 (chapter 7 in editions 1 and 2). A wiki [Vn02], the "Architecture made Practical" [Dr01] handbook, the presentation "NL Tax and Customs Administration Overview and Insight" [Di01], the old text "Viewpoints Functionality and Examples" [Lk01], and a lecture document by Knut Hinkelmann [Hi01] were also of assistance here.
Unfortunately, I was unable to find publicly available examples from practice.

[8] Source: ArchiMate 2.1 Specification (http://pubs.opengroup.org/architecture/archimate2-doc/chap08.html#_Toc371945239). Viewpoints are not a part of the standard since the version 3.0, yet they are provided in the "Informative" section of the standard document.

A view that would appropriately address concerns associated with the manageability of webservice interactions does not need all of the available concepts. When modelling an application landscape – i.e. all applications used by an organization – a certain level of detail should be omitted. In other words, we should use only those concepts of this viewpoint that are necessary for the insights we want to create.

To make this selection, I did not start with the language constructs of ArchiMate; rather, I turned to the real-life entities involved in service interactions, namely web services, interfaces and applications that provide and/or consume these web services. This is more appropriate, as we are examining the "entirety of all business applications"[9], thus, implicitly, the "as is" application landscape. Such a model requires verifiable knowledge, and it is beneficial to rely only on elements that are derived directly from observable entities.

Representing Applications and Services

At this point, more clarification is needed of the terms "applications", "services" and "service interactions" in the present context.

These terms are used informally, but in accordance with their common usage for real-life entities and their relationships. "Service" represents functionality exposed by a software component to other software components (in IT operations, we deal with service availability, mediation, endpoints and interfaces). "Application" refers to deployed software components (or a cluster of such components) that provide business functionality[10]. "Service interaction" denotes the dependency relationship whereby one software component invokes a service provided by another software component (i.e. one application provides a service and other applications consume that service).

Representing services and applications in ArchiMate is straightforward. ArchiMate provides Application Service and Application Component concepts that are the direct counterparts of real-life entities termed "services" and "applications", respectively.

Representing service interactions is slightly complicated.

In ArchiMate, there is no direct providing relationship between application components and application services.[11] The most appropriate way to deal with this is to derive the providing relationship by involving an *Application Function*. An *Application Component* is *assigned to* an *Application Function.* In turn, an *Application Function realizes* an *Application Service*. Both assignment and realization are structural relationships, and the assignment relationship is the stronger of the two. Therefore, if we omit the *Application Function*, we will derive the *realizes* relationship (for an explanation of derivation rules, see [Og03] and [Jo01]). In other words, the providing relationship can be represented by specifying in the model that an *Application Component realizes* an *Application Service*.

The consuming relationship is again straightforward: ArchiMate provides the direct "*serves*" dependency relationship between an *Application Service* and an *Application Component*.

Dependency chains

In practice, dependency relationships commonly propagate through the application landscape. For example, a portal application invokes a service implemented in the application that handles a specific business function. While processing that service request, the invoked application may need to further

[9] A subset of it: all applications that either consume or provide web services. However, this subset generally includes the most important business applications (BI excluded).

[10] There are issues associated with this, especially when we consider modern software components (e.g. Microsoft's API Apps), SaaS solutions in general and the DevOps trend. In my view, ArchiMate provides adequate constructs (e.g. aggregation and composition relationships), which can be used to address these issues.

[11] See the ArchiMate Application Layer Metamodel: http://pubs.opengroup.org/architecture/archimate3-doc/chap09.html#_Toc489946075

invoke several other services provided by other applications, such as the central CRM or a document management system. Consequently, the portal application depends not only on applications whose services it invokes directly but also on those applications that are being invoked while processing its service requests. In other words, the dependencies are "nested".

In the context of this article, for reasons of simplicity, we call these nested dependencies "dependency chains".

Applications, Services and the Boundary of the Organization's Application Landscape

Before going any further, it may be helpful to demarcate our perspective of the application landscape, in particular regarding services invoked beyond the organization's boundary. This concerns:

a) Services made available by partner organizations (external services) and consumed by applications within the organization's landscape;
b) Services exposed to the external world by applications owned by the organization.

The question thus arises: What should be included in our views and what should not?

Regarding external services, we should not be concerned with the providing applications. These applications are owned by a third party and do not need to be visible in our domain of concern.

Conversely, external applications that consume our services are most commonly of concern due to SLAs and other accountabilities. The service provider must guarantee the availability, functionality, service levels etc. to parties using the service, as well as restricting access to specific applications and/or users. These applications are hence "guests" in our application landscape and need to be included in views.

Exposing Services to Groups of Applications

We must deal explicitly with the (fast increasing) exposure of web services to unknown consumers[12] and/or mobile devices. Those consuming applications cannot be viewed in the same way as the rest, as the organization neither administers their individual connections nor has any other knowledge of them. However, the organization still needs to be aware of their service usage. As there is no agreed ArchiMate construct that provides for this, a language extension is required here [Og02].

I propose the following two alternative solutions:

A) Introduce two *Specialized Concepts* as extensions to the *Parent Concept* "Application Component":
   - "*Mobile Apps*",
   - "*Unknown Application Component*" to denote applications accessing the organization's unrestricted services.

B) Introduce extensions to the concept of "Application Interface".[13] An option would be:
   - "Human Application Interface"
   - "*Application-to-Application Interface*" to denote a general, non-human application interface with the following three extensions:
     - "*Mobile Apps Interface*" for different types of mobile applications.
     - "*Desktop Application Interface*" for applications executing on desktop computers.
     - "*Public Application-to-Application Interface*" for interfaces dedicated to accessing unrestricted services.

In the latter alternative, we could assume that the Application-to-Application Interface is a default (for non-human interfaces). If this default interface (one interface per service) is of no concern - which is mostly the case in our context, it could then be left out of the model. In other words, the list of interfaces in our view would be incomplete, in contrast to the list of applications and services.

[12] Some services are even dedicated to such access, e.g. in the case of "open data".
[13] Further information on Application Interfaces is present in the section "(Not) Representing Interfaces".

Chief Constituents of the Proposed View

In summary, to return to our main topic: we need to show both applications and services and their relationships, and (initially) no more than this. Two ArchiMate relationships, *Realization* and *Serving,* are sufficient.

Figure 2 Representing a dependency on a service

We may even opt not to show services. If we only need to show dependencies between application components, there is no need to include anything other than the application components (i.e. services can be hidden).

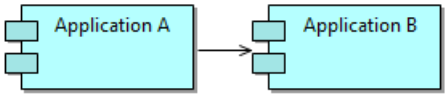

Figure 3 Representing a dependency without showing the service being used

The relationship between two applications is then derived. In this case, common sense deems it appropriate to use the *serving* relationship: *Application A serves Application B*. Of course, ArchiMate's formal derivation rules lead to the same result (see [Og03] and [Jo01]).

(Not) Representing Interfaces

In the context of web services manageability, we are concerned only with application-to-application interfaces, and not with user interfaces. Most commonly, a web service is exposed with only one interface and accessed via a single endpoint and there is only one binding to the underlying communication protocols. This makes the concept of *application interface* unnecessary for our purposes here, and we can simplify the application landscape view by omitting interfaces altogether. Even in the case of services with multiple interfaces, this should not be of consequence for the overall insights we are seeking. Here, of course, we assume that the concerns associated with multiple interfaces are addressed on another level.

There is a possible case for including interfaces for solutions where the same service is being exposed to separate groups of consuming applications via separate access points. Examples of such clustering are: internal vs external applications; mobile applications vs internal enterprise applications; and unknown applications (public interfaces for open data) vs known (internal or external) applications.

As discussed above in the section "*Exposing Services to Groups of Applications*", if interfaces need to be included in our view, we could extend ArchiMate and introduce the specialized concept of Application-to-Application Interface, as proposed in ArchiMate Specifications, 15.2.2, Examples of Specializations of Application Layer Elements (Informative) [Og2].

Dealing with Volume

A real-life application landscape includes hundreds or thousands of application-to-application connections. A "brute force" graphical representation showing the whole landscape would be unintelligible.

A solution would be to not use the visual language and instead show the integral landscape in tabular or hierarchical (tree) form, with only isolated parts of the landscape visualized graphically.

To make isolated parts meaningful, the real-life dependency-relationships need to be identified (either supported by an intelligent EA Tool automation or, painstakingly, by hand) and used as the criterion

defining segmentations of the application landscape. This is certainly a good approach, although there are two potential issues:

a) The number of elements that need to be shown may still be too large for a comprehensible 2D visualization;
b) The model constitutes a directed graph. As shown in Figure 4, nodes are *applications* and *services*, while edges are relationships: *serves* and *realizes*. However, this graph is generally not planar (the edges inevitably cross one another), which clutters the visualization and makes it less intelligible.

These issues remain a challenge.

Source of the Model Data

So far, we have not considered where the model data comes from. This is something that we need to do, as our view must be all-inclusive.[14]

It is unrealistic, in my opinion, to expect an organization to manually maintain an accurate, all-inclusive architecture model of its application landscape. To ensure a trustworthy overall insight, we need a mechanism that guarantees our model is a correct reflection of reality.

There are ways to collect the necessary information from the runtime application infrastructure and system administration tools. However, this is not an easy task, as data collection products are not yet standardized. The procedures involved require both discipline and an IT governance structure that enforces this discipline. Simply stated: an IT operations department is typically unable to immediately produce a list of applications and application-to-application connections currently in use. However, practical solutions to create architecture models from the runtime application infrastructure do exist, and some organizations have implemented these solutions without significant costs.

The problem of collecting information from the infrastructure is possibly aggravated by the increased use of cloud deployments, as business units may use applications and application services that are unknown to the IT operations department ("shadow IT"). On the other hand, the control over cloud deployments is commonly more formalized, which should make collecting information about applications and their interconnections easier than in the case of on-premise deployments.

To summarize:

a) maintaining an accurate architecture model of the application landscape is a challenge;
b) there is no standardized way to create a (adequate) model from the operational IT environment.

[14] One suggestion is to exclude less important applications. To do this, however, we first must identify them.

Visualization Example with Dependency Chains

Views shown in the examples were created using Archi®, a free ArchiMate modeling tool.[15]

The visualization displayed in Figure 4 is a real-life, anonymized[16] example showing the service dependency chain of a municipal portal application, "My City". The information was automatically derived from several API Gateways.

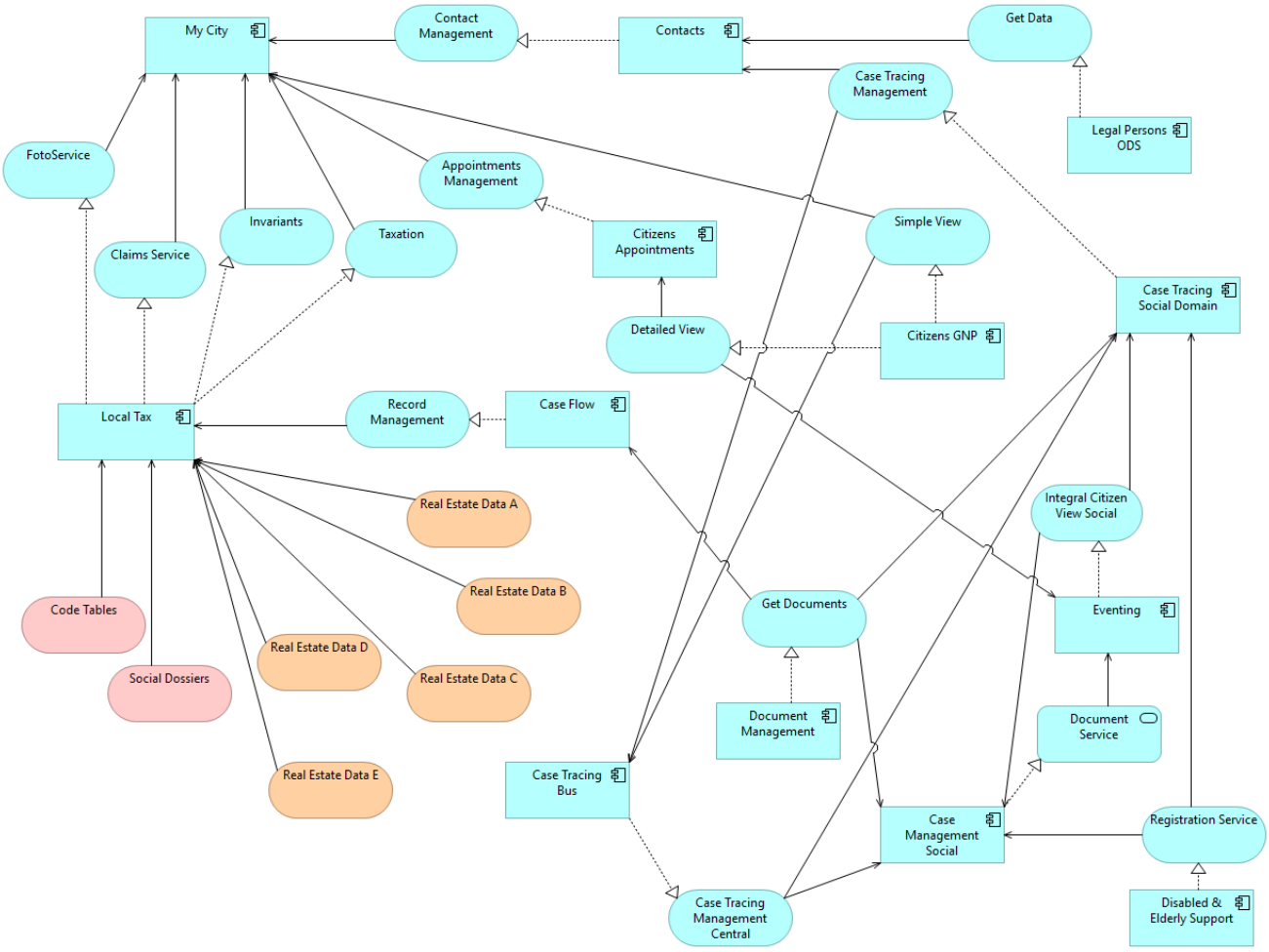

Figure 4 Real-life example (external service depicted in a different color)

This view has been simplified as follows:

a) The portal functionality is in reality provided by two web applications, and this model shows just one (*My City*). Viewing the dependency chains of these applications separately should not be a problem as there are no direct dependencies between the two;
b) Services provided by two other organizations are displayed in a different color. The providing applications of those external services are not shown (SOA best practice);
c) Only the services that are consumed in the dependency chain of *My City* are included in the view – that is, not all services of the visualized applications are shown.

Using this view, we can, for instance, check whether all service-providing applications match the availability level required by the portal application.

[15] https://www.archimatetool.com/
[16] All names are invented.

Visualization Examples with Mobile Apps and Open Data (using Language Extensions)

The following example shows the use of alternative shapes and colors for distinguishing mobile and unknown external applications.

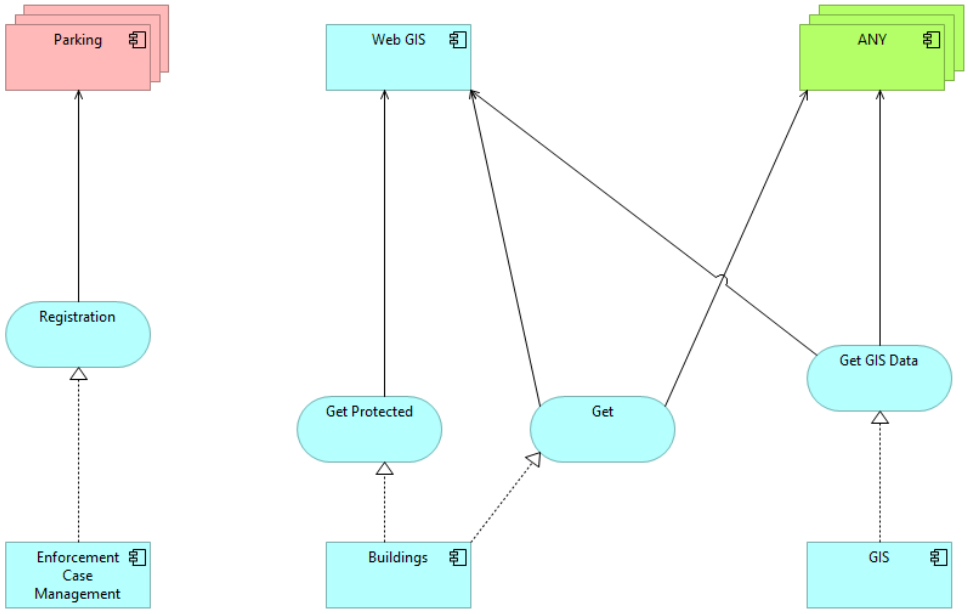

Figure 5 Alternative shapes and colors for extending the concept of ''Application Component''

An alternative to the above with interface extensions, as proposed in the section "*Exposing Services to Groups of Applications*", is shown in Figure 6. Different colors are used to depict the different types of application-to-application interfaces.

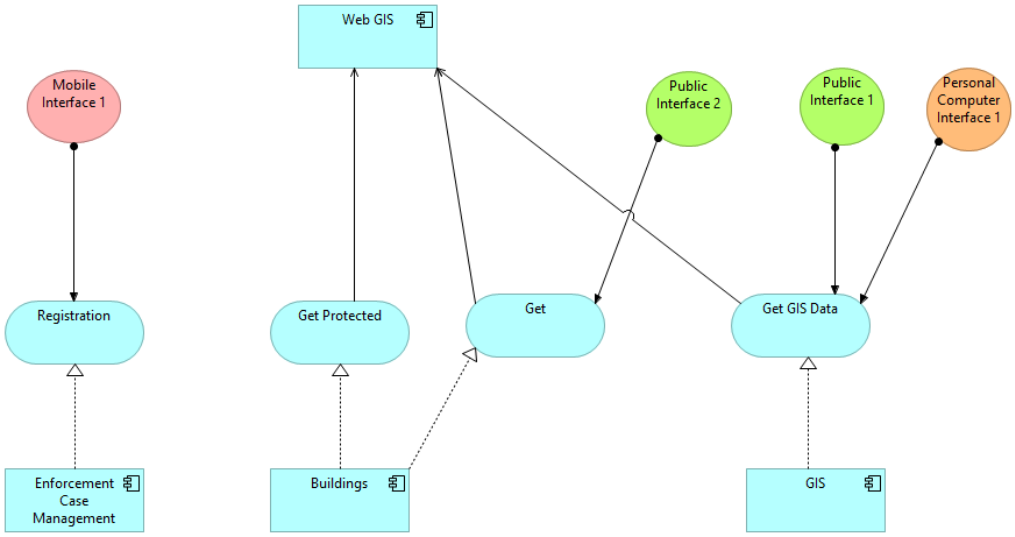

Figure 6 Using Application Interface extensions

Regarding the use of the associated service, the interface elements are the end nodes in this directed graph (please ignore the direction of the "assigned to" arrow); i.e. the consuming applications are not shown in this view. This is because the service-consuming applications are not registered individually.

Visualizations in 3-D

As mentioned previously, the visualization of *application services* and *application components* with their essential relationships "*serves*" and "*realizes*" forms a directed, non-planar graph. It is difficult to display

these graphs in 2D. Regardless of how cleverly the nodes are arranged, the graph is meaningful only when showing a limited number of nodes. And even then its comprehension requires some effort.

Given that we seldom use paper documents to examine application landscapes these days, there is no reason why we should not make use of 3D visualizations (especially following the widespread acceptance of HTML5).

The following figure shows a screen capture of a 3D visualization. It may appear unreadable, but the advanced features of 3D techniques can be utilized by using an appropriate display. Some of these are shown in a (teaser) prototype accompanying this article: http://www.soascape.org/prototypes3D/.

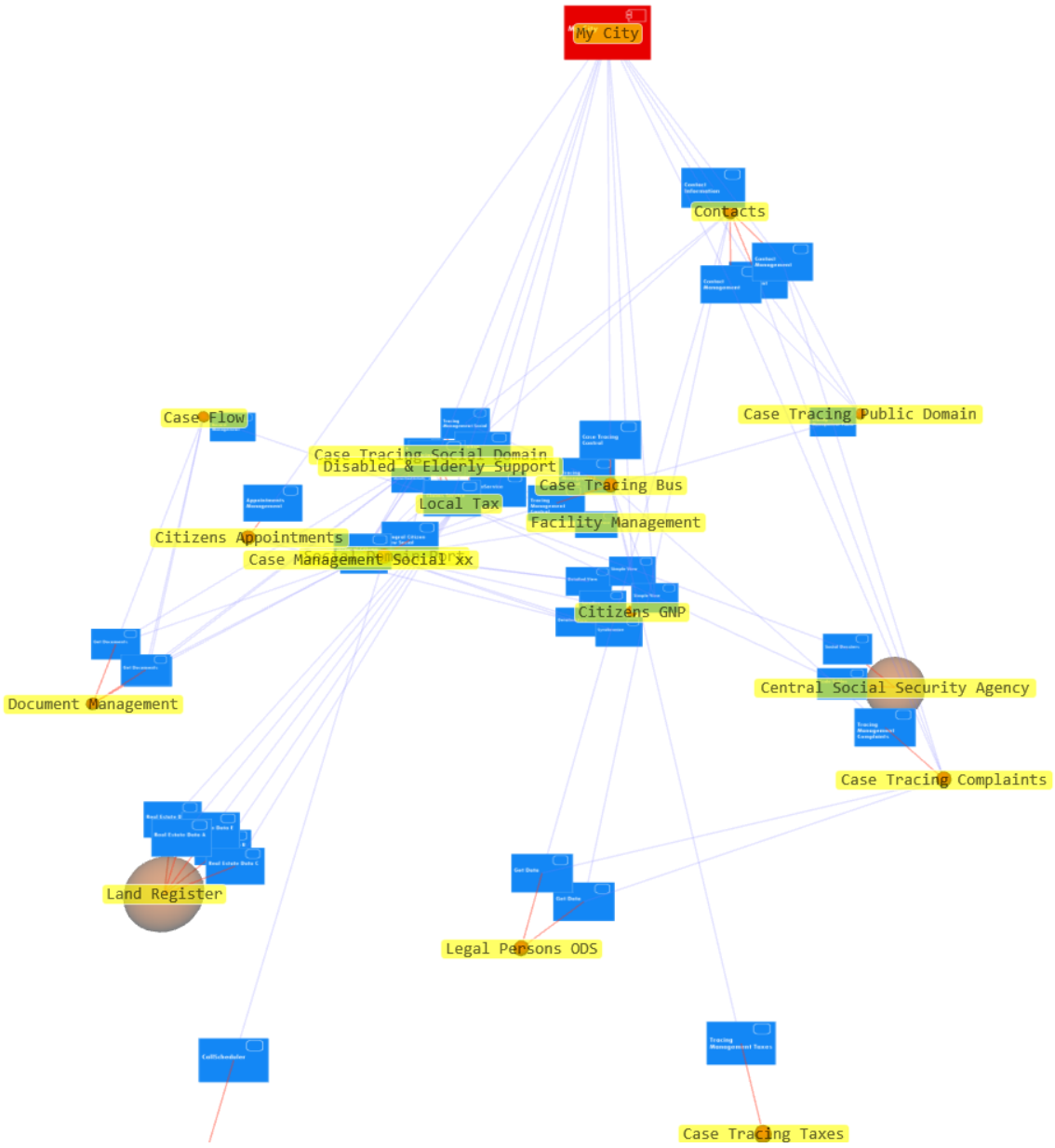

Figure 7 Example 3D visualization (a screen capture from: http://www.soascape.org/prototypes3D/)

Showing Dependency Chains

It should be emphasized that the "serves" relationship means a conditional dependency. Application components are "loosely coupled" and, in general, do not require all services to be available at all times. The decision about which services to invoke is governed by the business logic of the function being performed. For example, in Figure 4 we can see that the application "My City" uses the service "Contact Management" of the application component "Contacts". This component, in turn, may use the service "Get Data" of the component "Legal Persons ODS". Instances requiring this service invocation are very infrequent, and the portal "My City" can hence serve most users without being dependent on "Legal Persons ODS".

These dependency chains show the possible dependencies (services which may be invoked). This can be even more misleading when we consider that some services in the chain cannot, due to processing logic, be accessed in processes that start from "My City", as the application component may aggregate diverse functions. It may be argued that this is indicative of weak design; however, this objection is not relevant for the examination of "as is" application landscapes.

A more correct view of dependencies can be obtained via the ArchiMate concept of *Application Function*. However, this concept is an abstraction that does not directly correspond to a software artefact, which limits its usability for our purpose. Nevertheless, it can be used, if needed, to augment the observed (or automatically generated) information about services and applications.

Showing non-SOA Interactions

Is it appropriate to always examine web service interactions separately from the other types of application interactions (e.g. queuing, publish/subscribe, shared file locations, shared databases)? Could we show these in a same view?

My reason for omitting interactions based on these integration options from this paper is that they are far fewer in number than plain, synchronous web services. This is not to question either their importance or the volume of transferred data, merely to state that these interactions do not proliferate uncontrollably,[17] as is the case with web services. Consequently, they are less of a manageability concern, as they remain "on our radar".

This said, there is still a need for showing both webservices and other application-to-application interactions in one view, while keeping the distinction visible. This may be a challenge, as ArchiMate's "Application Layer" is mainly aimed at service orientation. An objection may be that if we model all interactions using the concept of services, we would be able to employ the Application Cooperation Viewpoint in the same way as we have done that for web service interactions. However, this would be a mistake – an erroneous attempt to give software solutions a meaning beyond that of their original construction. Models produced in such a way would generally do little more than cause confusion, as the concept of a "service" would be used inappropriately.

A standardized approach is therefore needed to show different types of application-to-application interactions in the same view, and the notation for this should not be limited to the concept of services. My belief is that the existing concepts in ArchiMate are sufficient for this purpose. Further discussion, however, is beyond the scope of this article.

[17] Interactions based on events and queueing could become as ubiquitous as webservices are today. The technical perquisites for this (standardization of queuing protocols (AMQP, MQTT) and standard product offerings, especially in the Cloud) are already in place.

Conclusions and Loose Ends

I began with arguments for the importance of including web service interactions in our views of application landscapes.

Subsequently, I have explored options of using the ArchiMate language to achieve this. I hope to have clearly established that the ArchiMate language provides a means for creating insights in dependencies caused by the large numbers of application-to-application interactions based on web services.

Several conclusions have been drawn. First, accurate insights involve a complete view of the application landscape. The underlaying architecture model must include all services, all providing applications, and all (or at least the significant) service-consuming applications. This leads to the next conclusion that dependency chains must be automatically identified by tooling and separately visualized, as necessary. If not, large numbers of applications and services would make visualizations unintelligible.

Maintaining the accuracy of a model that encompasses all applications and services is a problem unto itself. One option for keeping the list of services (and related applications) up to date is to extract the required information from the infrastructure (intermediaries such as API gateways[18] and ESBs).

However, information obtained in this way has its limitations. It does not include functional aspects, and may hence lead to incorrect conclusions about dependencies. An improvement would be to augment the information extracted from the infrastructure by using ArchiMate models that include information about application functions. Further exploration is beyond the scope of this article, as it would require a substantive study of ArchiMate relationships and their derivation.

There is no commonly agreed ArchiMate approach to representing service-consuming applications that cannot be dealt with individually. This concerns mobile apps, desktop applications, and "anonymous" external applications that consume services without access restrictions. I have proposed two alternatives, but other solutions undoubtedly exist.

2D visualizations have their limits, both in practice and in theory.

There are situations in which we should not isolate web services from other application-to-application interactions, i.e. we sometimes need to show all of those in the same Application Layer view of ArchiMate. In such cases, it would be useful if that view would also show the distinction between different types of application-to-application interactions. ArchiMate may well provide the means to derive such a view; nonetheless, I could not find publicly available sources that address this subject.

References

[Jo01] R. van Buuren, H. Jonkers, m. Iacob, P. Strating: Composition of Relations in Enterprise Architecture Models, 2004 (retrieved from Internet in June 2017, https://www.researchgate.net/publication/220713376_Composition_of_Relations_in_Enterprise_Architecture_Models)

[Bu01] S. Buckl, et al.: A Pattern based Approach for constructing Enterprise Architecture Management Information Models, 2007 (retrieved from internet in June, 2017, https://pdfs.semanticscholar.org/74b1/6d1f20db559dce8ffa9bb7ff6cbe340f5244.pdf)

[Di01] Marcel van Dijk, Lourens Riemens - NL Tax and Customs Administration Overview and Insight, a practical approach to connect business, applications and technology with ArchiMate (retrieved in July 2017, https://www.opengroup.org/public/member/proceedings/London-2016-04/archimate-case-study.pdf)

[Dr01] Hans van Drunen, Egon Willemsz (editors): ArchiMate made practical 4.0

[18] All popular API Gateways and ESBs expose management API's, which provide lists of configured web services.

[Hi01] Knut Hinkelmann “Enterprise Architecture Views and Viewpoints in ArchiMate - Reference” (retrieved in July 2017, http://knut.hinkelmann.ch/lectures/ABIT2016/ABIT%2004-3%20ArchiMate%20Views%20and%20Viewpoints.pdf)

[La01] J.F. Lankes: Metrics for Application Landscapes, 2008 (retrieved from internet in May 2017: https://mediatum.ub.tum.de/download/653232/653232.pdf)

[Lk01] Marc Lankhorst et al.: “Viewpoints Functionality and Examples” 2004.

[Lk02] Marc Lankhorst et al.: “Enterprise Architecture at Work”, fourth edition, 2017, ISBN-13: 978-3662539323.

[Og01] ArchiMate 3.0 Specifications, Example Viewpoints (Informative), 3.1 Basic Viewpoints in ArchiMate (retrieved from Internet in June 2017, http://pubs.opengroup.org/architecture/archimate3-doc/apdxc.html#_Toc451758125)

[Og02] ArchiMate 3.0 Specifications, 15. Language Customization Mechanisms (retrieved from Internet in June 2017, http://pubs.opengroup.org/architecture/archimate3-doc/chap15.html)

[Og03] 5.6.1 Derivation Rule for Structural and Dependency Relationships (retrieved from Internet in September 2017, http://pubs.opengroup.org/architecture/archimate3-doc/chap05.html#_Toc489946007)

[Vn02] VNA Wiki – Interfaces between applications (http://wiki.vianovaarchitectura.nl/index.php/Interfaces_between_applications)