The Systems Architecture Method

– An Overview

This document provides a very brief overview of the Systems Architecture Method (SAM). This is intended to provide users with sufficient methodological information and an understanding of the meta-model to configure and build their architectural models.

Understanding… The Meta-model

To use the Systems Architecture Method (SAM) effectively, securely and safely, it is essential that the user fully understands the fundamental structure of the meta-model, how the various diagrams and spreadsheets work and how the repository is populated and viewed.

In this section we describe only the salient features of each of these subjects.

Scope and Boundaries

All architectures and models must have boundaries and the scope of the work must be understood and applied. These boundaries define the “width” and scope defines the “depth” of our work. Some EA work can be described as “a mile wide and an inch deep” other projects tackle an area “an inch wide and a mile deep”. We need to have some reasonable compromise between these extremes. In SAM, we scope and bound our EA projects using the notions of domain, program, phase, context and timeframe.

We define these as follows:

**Domain:**

An industry, or governmental area of authority or a citizen-focused service, that is clearly understandable, comprehensive and largely self-contained from the points of view of governance, funding, management and professional competences. Examples might be banking, insurance, manufacturing, air transport, local government, education, law enforcement, healthcare and social care.

**Program:**

We define “Program” as a major coordinated endeavor aimed at revising or reforming some or all of the enterprise’s systems, organization, processes, infrastructure and technical platform within its domain. A program will encompass many projects in many disciplines over a period of time.

**Phase:**

Each program is divided, in a “timeline” sense, into phases each with a specified start and end point and defined criteria to judge success in their completion. Typically, these criteria are well-defined for the first phases of a program, but are less than firm for the later phases of the program. Often there is a formal “gateway” process to authorize transition from one phase to the next involving assessment of quality.

Highlight

Architecture often is described in terms of conceptual, logical and physical views. Conceptual views are high-level, are described using generalized terms and contain only the most important objects and relationships. A logical view shows the major architectural components and their relationships within the architectural boundary independently of the technical details of how the architecture is implemented. Physical views are the least abstract of the views of and illustrate the specific implementation components and their relationships. Sometimes, we might also add a contextual view which sets the general scope of the architecture.
achievement of objectives, costs, benefits and return on investment.

**Context:**

Each phase of a program can be expressed in differing perspectives of definition. We use the perspectives of “Conceptual, Logical and Physical”. Initially each phase is defined in conceptual terms and as work progresses the logical and eventually the physical definition emerge.

To these three contexts we sometimes add that of **Contextual**. This is an initial, almost “brainstorming” notion aimed at setting out the broad scope and boundaries of the program and its phases.

**Timeframe:**

Within each phase of each program, we need to define a progressive time element. Clearly, we recognize the current state as the “As-Is” and the ultimate end result as the “Vision”. In between we may have a number of “To-Be’s” each representing a step forward. Sometimes we have a “Has-Been” if we need to understand how we got to where we are.

We can now define some important terms we use in SAM:

- **Level 1 Combinations**
  - “Program/Phase” = Domain” + “Program” + “Phase”
  - “Architectural State” = “Context” + “Timeframe”.

- **Level 2 Combinations**
  - “Active Environment” = “Program/Phase” + “Architectural State”

- **Level 3 Combinations**
  - “Tree” = “Active Environment” + “Structure Code”
  - “Forest” = “Structure Code” + All applicable “Active Environments”

We should apply validation to these combinations. The principle is that each element of a combination must have been defined before you can define a new combination. In other words, you should not define an Active Environment simply by a random selection from Domain, Program, Phase, Context and Timeframe. You can only define an Active Environment from pre-defined, valid “Program/Phases” and pre-defined, valid “Architectural States”. This helps apply some degree of semantic integrity to our models.

**Schema Definition**

The default schema is shown in Figure 1. The schema is based on a network of structures and relationships. Each red sphere represents a “Structure” and each connecting line represents a set of potential relationships.

The default structures are:
- Organization (Departmental and Role-based)
- Business Processes
- Applications
- Information & Communications Technology
- Infrastructure & Locations
- Business Functions

© Systems Advisers Ltd, 2014. All Rights Reserved.
The Systems Architecture Method
– An Overview

- Data
- Components & Services
- Objectives & Goals
- Projects & Programs

The list of structures is not fixed and may be altered, added to or reduced to reflect individual situations and requirements. The only constraint is that the overall schema retains its integrity and cohesion.

The Overall Model

Hierarchies

A common way of organizing and categorizing a large volume of data is to form a hierarchy or tree, for example, as in a filing cabinet or a set of computer folders or directories. It is a convenient, and common, way of organizing data.

In the Toolkit, each structure is modeled by a “parent/child” hierarchy with up to six levels (although fewer will be sufficient in many cases); thus each entry (called a member) has a parent and a number
of children. For example, in the Organization structure, a Department (level 4 say) might have a parent, Division (level 3), and multiple subsidiary Workgroups (level 5). In some situations, a member may have more than one parent and this is modeled using a separate set off relationships.

A decision to be made is which summarization, or decomposition, scheme should be used to define the levels in the hierarchy. There can of course be more than one set of levels, each summarizing the base population of members in a different way. For example, the Organization structure may be decomposed on the basis of divisions and departments, say, or alternatively on the basis of roles and teams, or indeed both.

**Structures and Relationships**

**What is a structure?**

View it as a collection of information pertaining to a particular topic of interest, for example the enterprise’s ‘Organizational Structure’. Think of a structure as a set of filing cabinets containing all the information on a particular facet of the enterprise, filed and organized for easy access. Various filing schemes could be used but a major advantage would be to avoid redundancy. We only want to file a particular fact in one place not many. This makes it easy to find and easy to update. Also it would be advantageous to organize the information in a tree structure, or hierarchy, provided that it fits. Thus we can keep the detail at the bottom of the stack and have summary layers above, making it easier to deal with large volumes of information. The information need not be textual, it could include diagrams, documents (or their references), or multimedia items.

An enterprise’s ‘Organizational Structure’ may look like that in *Figure 2.*
This could be represented more generically by Figure 3.
In this basic illustration we have regarded the organizational structure as a number of levels. The levels are linked by a simple parent-child relationship between the members of neighboring levels. Each child has only one parent. This is of course a rather simplistic representation of an often complex structure. It ignores complications like matrix and project-based organizations. In a matrix organization one member may have more than one parent on the next higher level or perhaps have a parent on another level altogether. This is handled by the use of relationships which we will discuss below.

Changing a Structure
An Organization Structure is far from static however. Change is likely to be frequent and of two types. The first type – change to the value of a member is no particular problem – it is simply a matter of updating the ‘box’. The second type involves a more radical change to the structure – new levels can be introduced, restructuring can occur at a senior level, whole legs of the hierarchy can disappear and new ones grow. This may need redrafting of the specific structure but the generic form should survive.

More radical change does not usually happen all at once, it is usually implemented as of a change program with an ‘as-is’ and ‘to-be’ definitions and a migration program in between. Each migration step can be represented by a separate, parallel, hierarchy within the structure. We call these ‘trees’ and will define them later too.
Some structures may have qualitatively different facets too, for example, the conceptual, logical and physical aspects of particular objects. These too can be represented as parallel hierarchies within the one structure and this is a common feature of structures dealing with topics such as data.

Another Structure

Having thought about Organization, let’s now think about another structure – Business Functions – the things an organization does. Figure 4 shows a simple ‘functional decomposition’ – a hierarchical representation of the functions of the enterprise. This has been simplified greatly for explanatory purposes. In a real enterprise there might well be several hundred low level functional activities – called ‘primitive functions’ in some methodologies. These are defined in such a way as to be non-redundant, i.e. the same primitive function does not repeat in different legs of the hierarchy, nor do the primitive functions overlap in their scope. [Note the contrast with Business Processes in which the lowest level activities or tasks – sometimes called ‘elementary processes’ – are usually repeated, perhaps many times, in different processes. Incidentally, the lowest level in Business Function (Primitive Function) is the same ‘object’ as the lowest level of Business Process (Elementary Process), the differentiator is the absence or presence of redundancy within the hierarchy]

Figure 4 - Sample Business Function Structure

The generic model for Business Function might look like Figure 5.
The upper groupings – The Enterprise and its Functional Groups – contain information common to items at a lower level in the hierarchy. The middle layer contains information about individual business functions – description, purpose, operating parameters, etc. The lowest level – functional activities – would contain information about the particular tasks carried out within a business function. These represent the basic, indivisible units of work within the enterprise.

**Relationships**

An organizational unit could be said to have degrees of responsibility for, and involvement in, a business function. Therefore we could say that an organizational unit ‘is responsible for’, or ‘involved in’ or ‘interested in’ a particular function. Since the relationships are two-way, the reverse relationship may be expressed thus – a business function is the ‘responsibility of’, ‘involves’ or ‘is an interest of’ a particular organizational unit.

We can record these relationships using a spreadsheet too. Firstly, form a skeleton matrix. Take the Organization sheet and copy the organization hierarchy to the y-axis of the new spreadsheet and then add the Business function hierarchy to the x-axis, as in Table 1.
The Systems Architecture Method
– An Overview

Table 1. Skeleton Matrix - Organizational Structure vs. Business Function

<table>
<thead>
<tr>
<th>My Current Organisation</th>
<th>Planning</th>
<th>Marketing</th>
<th>Research</th>
<th>R &amp; D</th>
<th>Design &amp; Development</th>
<th>Manufacturing</th>
<th>Operations</th>
<th>Quality Assurance</th>
<th>Finance</th>
<th>Accounting</th>
<th>Human Resources</th>
<th>Legal</th>
<th>Review and Control</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>President</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vice President Finance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vice President Sales</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Order Control Manager</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electronic Sales Manager</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electrical Sales Manager</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vice President Engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R &amp; D Manager</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vice President Production</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plant Operations Director</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production Planning Director</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Facilities Manager</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purchasing Manager</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Division Lawyer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planning Director</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Relating Organization and Business Functions

Now let’s populate this matrix with the relationships between organization and business functions. However before we do this we have to think about the levels in the hierarchies at which we express the relationships. It is important to maintain consistency in expressing relationships and this is done by mapping at consistent levels between the two hierarchies in the matrix e.g. from level 3 in hierarchy A to level 3 in hierarchy B (or perhaps from 2>3 or 3>2). Which levels to choose is a question of granularity. Can I accurately express the set of relationships between the members of each level? Map too high, say at 2>2, and everything is related to everything else and there is no differentiation. Map too low, say at 4>4, and there is a very large number of relationships; perhaps too many to comprehend fully or at least define consistently in a short period of time. In our example, let’s try to map from level 3 of Organization to level 3 of Function. The result is shown in Table 2.
Table 2. Populated Matrix - Organizational Structure vs Business Function with Relationships at Level 3

Some ‘workarounds’ were needed to build this matrix:

- Dealing with a missing level in the hierarchy:
  - It will be noticed that ‘Personnel Director’, ‘Division Lawyer’ and ‘Planning Director’ report to the President but are not on the same level as Vice Presidents. We introduced a ‘dummy’ level-2 entry ‘Direct Presidential Reports’ to cope with this and keep relationships at the 3 to 3 level.
- Maintaining granularity:
  - We mapped relationships at level 3 to level 3. From this you can deduce the relationships at level 2 to level 2 and from them, level 1 to level 1. The higher level relationship is the same as the most senior relationship at the subordinate level.
The Systems Architecture Method

– An Overview

This also makes it easier to fill in relationships at a lower level, say at level 4 to level 4, since we have we have determined the relationships between the level 3 parents and thus can complete the lower level mapping in small pieces and perhaps do this over a period of time.

- Assigning seniority to relationships:
  - We have a choice of relationship value for each cell – R for ‘Responsible’, I for ‘Involved’, or T for ‘Interested’. These represent degrees of responsibility – R is the most senior, then I, and T is the most junior. There is also blank or ‘null’ value which indicates no relationship. You only need put one value in the cell; the senior ones encompass the juniors. Every column, at each level, must have at least one instance of the senior relationship.
  - Someone must be responsible for each function thus there must be an R in each column. Every row must have at least one value, i.e. must not be completely null – otherwise it has no relevance in the analysis.

Formalizing Concepts

So far, we have constructed two simple structures and shown how they may be related. The information in each of the structures was organized into a hierarchy, or tree structure, consisting of several levels of increasing detail. Each level contains items, or members, of a similar degree of detail or granularity. Each member has only one ‘parent’ on the next higher level and may have more than one ‘child’ on the next lower level.

Thus the basic organization of information within a structure is a tree structure. This is a very common approach in real life. However other kinds of structure are to be found, for example where one member has more than one parent on the next higher level. This is called a ‘network’ structure and may be found, for example, in a structure concerned with ‘Business Processes’. The levels within Business Processes will include one called Process and the lower one called Task.

However a common Task may be carried out in more than one Process, thus there is a ‘many to many’ parent/child structure between members on contiguous levels.

The relationships were formed between members of one level of the Organization tree to members of one level of the Business Function tree of similar granularity. Each relationship has a set of values which might be as simple as ‘yes or no’ or be more meaningful. In our example, an Organizational Unit was ‘responsible for’, ‘involved in’ or ‘interested in’ a particular Business Function. Relationships are bi-directional or commutative – a Business Function is the ‘responsibility of’, ‘has the involvement of’ or ‘has the interest of’ a particular Organizational Unit. A handy shorthand notation for a commutative relationship might be ‘responsible for<>responsibility of’. Each of these relationship values could have associated attributes such as effective dates.

We have been thinking about structures, their members and the relationships between them. This may be visualized in a very simple way in the shape of a “Dumbbell” - Figure 6.
We can develop this idea to illustrate the various artifacts we use in SAM – see Figure 7.

Table 3 offers definitions of each of these artifacts and we briefly discuss each below.

There may be multiple kinds of relationships between two structures. A bundle of relationships between two structures is called a ‘link’. Each kind of relationship should be semantically separate from the other relationships and describe a different notion. For example, if we had structures Cars and People, the link might include the relationships ‘owns<>owned by’ and ‘drives<>driven by’. Clearly each of these relationships could apply to a pair of members in their respective structures.

Sometimes members of a structure on a particular level have relationships between themselves. These are called ‘recursive’ relationships. An example may be Tasks within Business Process which
The Systems Architecture Method

– An Overview

have a ‘sequence’ within which they are executed. A recursive relationship of ‘follows<>followed by’ might be used to show this.

An important is the notion of versions within a structure. These may be regarded as parallel trees or hierarchies which describe differing aspects of the structure. These might for example represent the ‘as-is’ and ‘to-be’ aspects of the structure. This might be appropriate for a structure such as Organization when change is expected. Other trees might describe conceptual, logical and physical aspects of a structure such as Data or Application. A collection of related trees is called a ‘forest’.

Typically a user of an Enterprise Architecture is not interested in all of the structures in a model; they have a particular purpose in mind when exploring the EA which is very often linked to a job role and its specific informational needs.

Thus a useful aspect of an Enterprise Architecture is the capability of navigating selectively though a number of structures and links tracing through a set of meaningful relationships. We call this a ‘view’. In SAM architectures concerned with IS/IT, we particularly address Business, Application, Information and Technology views. Other frameworks have different views.
Table 3. SAM Structure Definitions

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure</strong></td>
<td>A structure is a body of information about a particular subject or topic of interest to an enterprise – structured, filed and organised for easy access. Typical structures in an IS/IT context might include organisational structure, business processes, locations, objectives and goals, data, applications and so on. An enterprise will have many such structures and on a wider non-IS/IT basis these might include markets, products, competitors, etc.</td>
</tr>
<tr>
<td><strong>Members</strong></td>
<td>A discrete piece of information belonging to a structure. A structure about ‘Locations’ might have the members ‘Head Office’, ‘London Sales Office’, ‘Birmingham Plant’, and so on.</td>
</tr>
<tr>
<td><strong>Level</strong></td>
<td>The members within a structure may be divided into groups which recognise different summarisations of the information. The criteria for a particular kind of structural group may include increasing levels of detail about the members. Each of these groupings of members can be represented as a level within a hierarchy of information. Typically such hierarchies are formed using a single parent/multiple child relationships – a tree structure (decomposition) or a multiple parent/multiple child relationships – a network structure.</td>
</tr>
<tr>
<td><strong>Links</strong></td>
<td>Structures are related to each other. Between two structures there may be many connections with different meanings. For example a structure ‘Vehicles’ and a structure ‘Party’ might have connections like ‘ownership’, ‘driver’, ‘knocked down by’, ‘insured by’ etc.</td>
</tr>
<tr>
<td><strong>Relationship</strong></td>
<td>A connection between two members of different structures that expresses a link. There may also be relationships between members of the same structure (recursive relationships).</td>
</tr>
<tr>
<td><strong>Trees</strong></td>
<td>A tree is a hierarchical organisation of members of a structure into groups with a different focus such as conceptual, logical or physical characteristics or temporal characteristics (past, present and future). A tree contains the full hierarchy of levels of the structure, i.e. it may be viewed as a vertical ‘slice’ through the structure.</td>
</tr>
<tr>
<td><strong>Forest</strong></td>
<td>A collection of Trees, the set of which makes a coherent ‘family’, e.g. conceptual, logical and physical trees.</td>
</tr>
<tr>
<td><strong>View</strong></td>
<td>1. A representation of a whole system from the perspective of a related set of concerns (IEEE Std 1471-2000). 2. A meaningful collection of information composed from corresponding trees of levels from different structures and their relevant relationships. (SAM)</td>
</tr>
<tr>
<td><strong>Viewpoint</strong></td>
<td>1. A specification of the conventions for constructing and using a view. A pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis. (IEEE Std 1471-2000). 2. A template for the construction of a specific view from a set of structures, their levels, trees and the associated relationships. (SAM)</td>
</tr>
</tbody>
</table>

In formulating SAM some years ago, we were impressed by the form of the Atomium “building” in Brussels, Belgium. See Figure 8 - The Atomium Figure 8. Our impression, quite unfairly on the
The Systems Architecture Method
– An Overview

architect, was that it resembled an assembly of “dumbbells” – just like our overall EA model. Since then we have used the analogy to explain the model and its navigation.

Figure 8 - The Atomium

The analogy with Enterprise Architecture may not be immediately obvious. As we have said, in system architecture work we are concerned with discovering facts about the enterprise, and the problem domain, and understanding how these facts relate to each other. From this understanding, we deduce various strategies and initiatives to modify and extend the facts, hopefully to the benefit of the enterprise.

The Atomium building is a useful illustration of the idea, which is the basis for SAM, in which we use the notion of structures, or “spheres of interest”, to represent coherent groups of facts, and the notion of the connecting tubes to represent the relationships between the groups of facts.

Placing ourselves, metaphorically, in one of the spheres, we can examine the facts in our sphere of interest, and then, look down the tubes and see related facts contained in another sphere of interest. In SAM, we might even travel along a tube (the Atomium building actually has elevators and escalators within some of the tubes) and go on a voyage of discovery from one set of facts to another, and even onwards to further related sets of facts.

1 Photo: R J A Jarvis

© Systems Advisers Ltd, 2014. All Rights Reserved.
Members and Identification

The “member” is the fundamental object in our architectural model. Each structure has a population of “base members” that form the lowest level of the structure hierarchy or tree and are indivisible objects that do not decompose any further. Members populate all levels, of course, those at higher levels being the summarization members for the level below.

The member population does not need to be complete, and indeed, rarely is. It is sufficient to stop the top-down decomposition at the level at which meaningful relationships between groups of members in different structures can be formed. This is usually at about the third or fourth level from the peak of the hierarchy. Further having reached the useful level, it is often only necessary to “drill down” further for those members in the immediate area of interest.

We record the information for each member in a common format irrespective of the structure and level to which it belongs. In summary, this is:

- Member ID and Revision Level
- Structure Code, Level Set and Level
- Member Short Name
- Parent Member ID and Revision Level
- Member Description
- Member Attributes (up to five, user configurable by structure)

In the repository we hold all member records in one table. This, and the common format, enables a highly flexible approach to analysis.

Figure 9 shows a snapshot of the Structure Members table in the Toolkit Repository. The formatting of the Member ID should be noted.
 Links and Relationships

Structures are, of course, associated with each other in many ways. For example, Applications use Technology, provide Business Functionality and manage Data. Business Processes use Applications and are carried out by Organizational units. Objectives and Goals are realized by Projects and Programs and so on. A link between two structures can have many semantic meanings each of which can be represented by a carefully defined relationship. Each of these relationships is expressed using a descriptive phrase such as those above.

Relationships may be declared between the members of one level of structure A and the members of a level of structure B. Thus we could record that a department (Level 4 in the Organization structure) is located in a certain Building (Level 3 in the Infrastructure & Locations structure).

It is the mapping, analysis and interpretation of relationships that engenders knowledge and understanding of the enterprise. Many times users have a sudden revelation as they map out a set of relationships. “Ah, that explains it!” is commonly heard as the explanation of a business issue is revealed.

A particularly important relationship is that a Business Function “creates, reads, updates or deletes” Data. The CRUD relationship, as it is known,
The Systems Architecture Method

– An Overview

describes which business activities operate on the enterprise’s data resource. The construction and analysis of the CRUD relationship is a key activity in data-centric analyses such as in Service-oriented Architecture.

The best way to map relationships is to form a matrix with the structure members on the axes and the relationship values in the cells. A spreadsheet is a good vehicle for the construction of the matrix and Figure 13 shows a portion of such a matrix in Excel.

Cluster Analysis

Although many insights emerge as the basic matrix is constructed and studied, it is the subsequent manipulation and remapping of relationships that brings most return. A particularly useful technique is that of “Commutative Clustering” - a major feature of the SAM method.

The technique reveals patterns of relationships in the matrix which can be of considerable value in the discovery and understanding the underlying truths of the enterprise. In simple terms, Commutative Clustering groups together pairs of members which are related by common ‘clusterable’ relationships such as Create and Update (CU) in the CRUD matrix. Slightly inaccurately, this may be described as a “double sort” of the matrix, firstly by a column and then by a row, progressively moving the origin down a column and along a row and sorting again.

We have defined a procedure for manual Commutative Clustering. We call this the “North West” method and it is detailed in Figure 10.
As an example let us cluster the matrix of Business Functions and Organisation that we built earlier using the North West method. If we cluster on the relationship value 'R' – Responsible for – at level 3, the resulting clusters will contain all organisational units responsible for particular business functions and also all business functions that are the responsibility of the particular organisational units. The clusters may be call ‘Responsibility Groups’. See Figure 11.

This result, one big cluster and five small ones, is a bit disappointing but not untypical of a first pass. A picture like this is caused by dubious relationships that actually ‘join’ clusters together. Is the Purchasing Manager really responsible for cost planning? Probably not, it which case we can downgrade the relationship to ‘involved in’. Similarly, is the Engineering Design Manager only involved in Design and Development? We should probably upgrade the relationship to ‘responsible for’. Adjusting the dubious relationships gives the result in Figure 12.
# The Systems Architecture Method

## An Overview

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>I</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Purchasing Manager</td>
<td>R</td>
<td>I</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Division Lawyer</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Planning Director</td>
<td>I</td>
<td>I</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>R</td>
<td>R</td>
<td>T</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Treasurer</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td>R</td>
<td>T</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Plant Operations Director</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>R</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Production Planning Director</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Facilities Manager</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Materials Control Manager</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>R</td>
<td>R</td>
<td>I</td>
<td>T</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Order Control Manager</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Marketing Manager</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>R</td>
<td>I</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Electronic Sales Manager</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Electrical Sales Manager</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>R &amp; D Manager</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Engineering Design Manager</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>I</td>
<td>I</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Personnel Director</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>T</td>
<td>R</td>
<td>T</td>
<td></td>
<td>R</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
</tbody>
</table>

**Figure 11 - 1st Pass Clusters**
We used this clustering technique to good effect in our Healthcare Business Pattern to define the business components and services. *Figure 13* and *Figure 14* show a portion of the Healthcare Business Pattern clustered matrix in both “raw” and “processed” form. Candidate business components are highlighted in yellow. The original matrices can be viewed in the Solutions Toolkit.
The Systems Architecture Method
– An Overview

Figure 13 - CRUD Matrix (portion - unprocessed)

Figure 14 - CRUD Matrix (clustered - portion)
Granularity and Volatility

The population of structures and their associated relationships can be very large. A typical structure may well have over 10,000 members in total over all levels and a set of relationships, formed at the level 6 of the hierarchy, could well exceed 100,000 entries. This clearly is a problem in so far as the effort required in data capture and validation is equally large. However at level 4, a typical structure may have less than 350 members and relationships to another structure at this level might total between 1,000 and 2,000 entries. This would clearly be a more appropriate level, or indeed level 3 might be too, at which to define relationships provided that the summarization scheme is such that the relationships declared are meaningful and truly encapsulate their subordinate levels. It usually is!

Context and Timeframe

The data we record and analyze in an Enterprise Architecture needs to be clearly identified and consistent in terms of its context and the timeframe to which it refers. With regard to context, architectural data may have a conceptual, logical or physical semantic meaning. With regard to timeframe, the data may refer to the present state (“as-is”), the desired end state (“vision”) and a variable number of intermediate, in-progress states (“to-be” stages). Each model, object, relationship and definition in the architecture should have a clear context and timeframe e.g. “logical, to-be” or “physical, as-is”. We call this “Architectural State”.

Forests and Trees

We have described how we can organize structure members into a hierarchy or tree structure. We have also noted that all data should have context and timeframe. We handle this by forming a tree for each context and timeframe combination (Architectural State) within a structure. Thus for each structure, there is a family of trees that track the evolving population from “as-is to vision” on one hand and progressive development from “conceptual to physical” on the other. We call this collection of trees for a single structure a “Forest”.

Views and Viewpoints

The individual users of an Enterprise Architecture have differing purposes and motivations. Some are interested in the business aspects of the architecture – in Objectives and Goals, Projects and Programs and Business Processes for example. Others have an interest in technical matters – Technology and Infrastructure for example. Each of these requirements can be met from the same Enterprise Architecture. Since the members, structures, forests, trees and relationships are common to all views, there is the assurance that all views are consistent being drawn from the same population in the same level of update.

Four popular views are the Business, Application, Information and Technology views (BAIT for short) and these are fully supported in our architectural model.

The BUSINESS VIEW describes how the business works. It includes broad business strategies along with plans for moving the enterprise from its current to its future state (as-is to to-be).

The APPLICATION VIEW defines the enterprise’s application portfolio. Typically it would be based around the application structure represents the services, information, and functionality that crosses organizational boundaries, linking users of different skills and functions to achieve common business objectives.
The Systems Architecture Method

– An Overview

The INFORMATION VIEW describes the data the enterprise needs to run its business processes and operations. The information view may describe how data is bound into the work flow, including structured data stores such as databases and unstructured data stores such as documents, spreadsheets, and presentations that exist throughout the enterprise.

The TECHNOLOGY VIEW lays out the hardware and software supporting the enterprise and provides a logical description of infrastructure and system components that are necessary to support the application and information views. It defines the set of technology standards and services needed to execute the business mission.

An important point is that these views, and others, are drawn from the same members, relationships, structures, trees and forests. In other words, the views are consistent with each other and data mismatches, and resulting bad decisions, are avoided.

The Minimum Essential Models, described later, are also views drawn from the same, consistent data as are the key architectural scenarios in our Business Patterns.

Environment Definition – Scope and Boundaries

The development of an Enterprise Architecture is not an unconstrained activity without limit to its range and depth. It is normally limited to the boundaries of the enterprise it describes. Nor is it a parochial activity, limited to a couple of departments or even a division of the enterprise.

An Enterprise Architecture must address a coherent and cohesive business environment and the issue is how this should be defined. We suggest the following parameters may be used in setting the scope and boundaries of any EA project:

Domains: The architecture should be completely contained within a business domain such as healthcare or social care. Any overlaps between domains in terms of functionality or data usage should be handled by means of a common domain.

Business Programs and Phases: The architecture should address one or more complete business programs and all their phases. By business program we mean programs of work aimed at business development or improvement and involving multiple business functions.

Business Processes and Data Scope: The scope of the architecture should include complete business processes and all data creating and updating functions within the domain.

These parameters now provide the definition and identification of a “tree”. The Tree ID is composed from the following elements:

- Program Phase (Domain + Program ID + Phase Code)
- Architectural State (Context + Timeframe)
- Structure Code
- Tree Description
- Forest Name

Highlight

There may seem to be an apparent contradiction here in so far as we seem to be describing large scale enterprise architecture projects even though our toolkit is aimed at the single architect or analyst. However, even small scale projects should be governed by these principles.
The Systems Architecture Method

– An Overview

We allocate a member to a tree. Since members are held in a “pool”, we can assign a member to more than one tree. For example, a member may be unaltered between “as-is” and “to-be” states and thus is included in both trees. It is the analysis of “tree populations” that provides the basis for migration planning in that it shows which members are new, which are unchanged and which are discontinued. The differencing of two trees (the “delta”) forms an important input to change management programs, e.g. it specifies the changes in Organization or Business Processes, Technology, and so on. This is an important usage of Enterprise Architecture and is a highly useful tool for even for the single architect user.

The Myth of Enterprise-wide, Project-deep Projects

Have defined our scope and boundaries, it is not reasonable to expect an Enterprise Architecture to specify all levels of detail from business processes through technology selections and application functionality for individual projects (project-deep) across the entire enterprise (enterprise-wide) in one massive and collective effort.

Yet, this is what many Enterprise Architecture projects attempt. They use armies of architects and consultants who closet themselves away for months at a time and then deliver “the answer.” The problem with this approach is that the answer is usually out of date by the time it is delivered. In attempting to define all things to all people, this approach severely compromises the value of any results.

Our development method recognizes these limitations and takes appropriate measures to build the Enterprise Architecture in successive iterations. This allows the architecture to provide business value quickly, to gather feedback from actual use, and to make adjustments through subsequent iterations. Following the initial iteration you can state that you “have an Enterprise Architecture.” However, the moment this is stated as a fact, the work is just beginning.

The iterative process is supported by the concept of the Minimum Essential Model. It is only necessary to build the part of the model needed for the immediate problem in hand and only to the depth required to express meaningful relationships between the Structures Enterprise Architecture. Then you can move on to the next MEM.

Controlling the Project

We need to translate the scope and boundary definitions into a clear plan. This need not be a complex affair. We need to establish our priorities before embarking on a project, in either Explore Mode or Enterprise Mode, and establish our deliverables, timeline and resource requirements.

As a single user project, we can use simple project methods – perhaps a basic Gantt chart is enough. Beyond that, when working on a multi-architect project, the standard project management procedures and tools of the enterprise should suffice.