A Network-based View of Enterprise Architecture

Bala Iyer	David Dreyfus	Per Gyllstrom
Information Systems Boston University School of Management 595 Commonwealth Avenue, Boston, MA 02215	Information Systems Boston University School of Management 595 Commonwealth Avenue, Boston, MA 02215	Principal Architect Component Technology & Architecture Group (CTAG) PFPC Worldwide Inc. 4400 Computer Drive Westborough, MA 01581

Traditional notions of architecture have focused on the components (or domains of interest -- process, data, and infrastructure) aspects of architecture. Their goal is to separate concerns into modules and provide interfaces between modules. This view helps designers understand the ideal or espoused view of architecture. In our work we view architecture from a dependency perspective. These dependencies evolve over time, creating an emergent architecture. The emergence is influenced by both technical and social factors. Dependencies occur during the design, production, and use of enterprise components. This leads us to use network-based analysis techniques in order to understand the emerging dependency networks.

In order to provide architects with support tools to communicate and make decisions about architecture, we describe the data requirements and algorithms that can be used to build a decision support system that enables enterprises to incorporate a network perspective in their decision making process. We present our approach and methods in the context of a case study.

Key words: Architecture, dependencies, networks, emergence

Introduction

A fundamental assumption behind this work is that reasoning with architecture is a key determinant of successful information systems design, maintenance, and evolution. Stakeholders need architectural representations in order to make resource allocation decisions and perform risk assessment. In addition, we believe that current techniques fall short in providing stakeholders with a vehicle that helps them make informed decisions about information system design. Part of the problem is that IS architecture has no universally accepted definition in either the research arena or in the practitioner world (Ross, 2003). Architecture has been viewed strategically (Henderson & Venkatraman, 1993; McKay & Brockway, 1989; Morris & Ferguson, 1993; Ross, 2003; Sauer & Willcocks, 2002), organizationally (Byrd & Turner, 2000; Dreyfus & Iyer, 2006; Duncan, 1995; Iyer & Gottlieb, 2004; Richardson, Jackson, & Dickson, 1990; Weill & Broadbent, 1998), and technologically (Malone & Crowston, 1994; Messerschmitt & Szyperski, 2003; Nezlek, Jain, & Nazareth, 1999; Parnas, 1972). These perspectives, while individually interesting, do not provide the integrated view of architecture that is required to analyze risk and make resource allocation decisions. Another part of the problem is that these perspectives often focus on the idealized system, and not the system in use.

Previous studies have summarized that an IS architecture includes a group of shared, tangible IT resources (i.e., hardware, software, data, training, management, etc.) that provide a platform to launch present and future business applications (Duncan, 1995; Kayworth, Chatterjee, & Sambamurthy, 2001; McKay & Brockway, 1989; Weill & Broadbent, 1998). Architecture, as implemented through its IT infrastructure, should be flexible, reliable, robust, scalable, and adaptable (Byrd & Turner, 2000; Duncan, 1995; Kayworth, Chatterjee, & Sambamurthy, 2001). It should support the reuse of business components within a firm, while supporting firm responsiveness, innovativeness, and economies of scope (Kayworth, Chatterjee, & Sambamurthy, 2001). A review of these and other articles clearly illustrates that when the given definition for architecture is translated into action, the concept becomes very complicated.

Architecture implementation also involves learning effects. As researchers have explored organizations making changes to architecture, they have identified two strategies: localized exploitation or enterprise-wide integration (Allen, 1977). Most techniques seem to support the former and there are many tools that support the latter.

These techniques and tools, however, focus on the system at a point in time. Changes to the system, on the other hand, do not occur in one step; they occur in stages (Ross, 2003). Although seemingly obvious, these insights don't seem to have been translated into tools or techniques.

Architecture is more than technology. It reflects and supports business strategy. Architecture is not just concerned with the allocation of resources at the physical level, but also with the support of strategic business goals. The architectural challenge is not just cost minimization in the allocation of task to computational device, but the alignment of the task structure (Gasser, 1986), supported by the information system, with the business objectives of the organization. Thus, the approach we take to architecture must enable communication and decision making between and by business and technology stakeholders.

Zachman provides a useful framework that identifies the components of an IT architecture as well as the various perspectives taken during the design and implementation of an architecture by the different stakeholders (Zachman, 1987). According to Zachman, there is no such thing as a single information architecture; there are many. Separate architectures exist for scope/objectives, business model, information system description, technology model, detailed description, and machine language description. For these six categories, there are also the descriptions for *who, what, how, where, when,* and *why*. Altogether, there are thirty-six possible architectures. This insight liberates us from the constraint of a single architectural perspective.

D'Souza and Wills describe the architecture of a system as the set of design decisions that constrain its implementation and maintenance (D'Souza & Willis, 1999). They discuss the need for many different architectural views of a system, each using a different set of elements (abstractions), each conveying significant design decisions. Some common architectural views focus on technology deployment (hardware, networks), processes, software packages and their structure, object types and relationships.

Iyer and Gottlieb (2005) identified three views through which we can examine architecture. The first view is the espoused (Architecture-in-Design). It is the outcome of the process of defining and modeling the architecture and describes the planned dependencies between system modules. They argue that although there may be multiple participants in the design process, the espoused view is the province of the IS Architect. This is the view we most commonly associate with architecture. In reality, most if not all enterprise architectures are the result of individual "silo" application implementations or applications brought into the enterprise though mergers and acquisitions, resulting in an enterprise architecture that likely will differ from the espoused view. This emergent architecture is the second view. It is a descriptive view of the actual dependencies that exist among system modules. It is the system as it exists at a point in time. Finally, Iyer and Gottlieb define the third view, architecture -in-use (Architecture-in-Operation). The in-use view highlights the dependencies between and among system components and organizational groups that arise from the business of doing the work of the enterprise – selling products, buying supplies, managing employees, etc - as employees, suppliers, and other stakeholders interact with the system.

The emergent architecture, the focus of this chapter, can be a result of conscious or unconscious action (Alexander, 1964). That is, organizations with information systems can evaluate projects with the intent to understand how the project will impact the emergent architecture, or they can evaluate projects without this architectural perspective. An organization can develop an architecture, implement it according to plan, and then have a series of subsequent IT projects that, in response to specific or general business requirements, modify it. Iyer and Gottlieb (2004) argue that the way architecture emerges may subsequently impact how well the firm can achieve IT/business strategy alignment as well as the goals of flexibility, reliability, robustness, scalability, adaptability, and reusability.

In the quest to align IT and business strategy, many approaches are relevant. Take, for example, the technical perspective taken by architecture modeling (Zhao & Siau, 2002). UML modelers develop use cases from which models are developed, structured analysis highlights the relationships between data elements and processes (Yourdon & Constantine, 1986), and Petri-Nets have been used to analyze coordination constraints (Gunter, 2000).

Most of the approaches described so far have focused on the components. In some cases they focus on data and process independently, or, as in the case of objects, they encapsulate data and process, but deal with them as a combined independent entity. These approaches are useful to understand espoused or intended architecture when we are interested in static systems. However, most systems deal with dynamic environments and the accompanying changes in requirements. These changes affect both individual components and their dependencies with other components. As a result, we have to focus on the direct and indirect (cascading) dependencies as well. Network analysis techniques are a natural fit for this task.

A network approach enables us to conceptualize large-scale information systems and their emergence over time. Through the techniques and concepts we will present in subsequent sections, architects can better communicate across stakeholder groups, better cope with emergent complexity, and better manage risk.

Network approach

Central to our approach to architecture is representing information systems as a network. All renderings of actual information systems are abstractions: in our case information systems are drawn as networks comprised of nodes and links. In particular, a node is a component with a well-defined interface. As such, it may be used by other components without the user of the component being aware of its implementation details. Ideally, the component is separately buildable, installable and manageable. In practice, it is hard to substitute one component – even if part of the same product class – for another.

The links between nodes represent conduits through which information, ideas, and knowledge flow. In the case of information systems, the most obvious links between systems are those in which two systems exchange data. That is, two nodes are linked if they share data. As a dependency, we would say that system A depends on system B for data. However, by drawing on the coordination literature (Malone & Crowston, 1994) and organizational research on interdependence (Thompson, 1967), we understand that there are multiple types of dependencies by which we could characterize the tasks represented by systems: resource sharing, producer/consumer, and simultaneity. Links between nodes are limited to data flows in this chapter, but they can be expanded to include a great variety of dependencies that are found in practice.

A resource dependency exists when system A depends on data managed by system B. For example, applications that share a common Oracle database have a dependency on the shared data resource managed by Oracle. The database management system (DBMS) has many facilities to manage the dependency on this shared resource. A producer/consumer dependency exists when system A depends upon the results of system B. For example, a general ledger application depends upon the data processed in a sales order application. A simultaneity dependency exists when two or more tasks must be completed before a third task can begin. For example, a reporting application may need data from multiple systems before compiling the report.

Network analysis has been utilized in a great many areas to describe many complex phenomena (Barabasi, 2002). Software researchers have proposed metrics (Chidamber & Kemerer, 1998) to assess the structural properties of software and measure complexity using techniques such as DSM (MacCormack, Rusnak, & Baldwin, 2005). In our approach, we are introducing the concepts of social network analysis into the examination of architecture because information systems are both the product of social construction (Pinch & Bijker, 1987) and are first-class actors in the network that includes the stakeholders of the information system (Akrich, 1992). It is imperative that the architect understand how the information system fits within the larger network that is the organization. The computer-based information system is the manifestation of an ongoing dialog between the different stakeholders of the system. It is both shaped by the organization and will shape the organization in a cyclic, recursive process (Glaeser, Kallal, Scheinkman, & Shleifer, 1992; Orlikowski, 2000). Therefore, the information system architect plays a role in shaping both the information system and the subsequent organization. Changes made by business managers shape both the organization and the subsequent information system.

Enterprise architects are interested in such variables as IT project complexity, cost and success, as well as system flexibility, robustness, adaptability, and performance. They are also responsible for the ongoing evolution of the enterprise architecture. These outcomes are the result of an interaction between system and organization that system architecture influences (Glaeser, Kallal, Scheinkman, & Shleifer, 1992). Project cost and maintenance is directly related to complexity (Banker, Datar, Kemerer, & Zweig, 1993), which is a description of interdependency. If the goal of architecture is to provide flexibility, robustness, and adaptability, then any change to the architecture will affect those characteristics.

Managers looking to address IT project performance and overall system performance must balance the needs of various internal and external organizations, and even engage in the negotiations that decide whether a system is a success or failure (Wilson & Howcroft, 2002). Dependencies between components of the IS architecture reflect dependencies between the organizations that utilize the components. The architect's ability to modify the network is constrained by the interdependencies of the groups utilizing the existing components. Moreover, the interdependencies evolve as the information system is adapted by the organization, and as the organization adapts to the information system. Through frequent intervention, the architect can influence the evolution of the information system, but may not be able to control it.

If we assume that the architects cannot control all the components of the information system, then the question the architect wants answered is which components should be controlled, or more actively managed. That is, what are the architectural control points? Similarly, how does the espoused architecture influence subsequent emergence and control points? Are there architectures and simple rules that minimize the change in control points? The control points may be identified according to a specific set of criteria (i.e., their autarky value), or they may be identified due to their network position within the espoused architecture (i.e., their positional value). In a system of emergent interdependencies, however, the systems identified as core by one set of managers, at one point in time, may not be identified as the most important by either the original set of managers or other groups of stakeholders at another point in time. That is, the loci of interdependence may shift over time. In our approach, we identify and emphasize importance by network position, as described later, in order to complement traditional approaches. The network approach in no way minimizes the need to actively manage systems with high intrinsic value.

New measures and visualizations

Networks provide two benefits to understanding architecture. First, they provide a good metaphor to communicate architectural issues. One of the challenges architects face is that architectural models and specifications are often useful only to other architects. Enterprise architects, chartered with a long-term, global architecture strategy, are often challenged to convey how they contribute to the day-to-day business. This includes conveying what the key issues are, and why projects get out of control (fail, run over budget, ship late, fail service level agreements, etc.) to their business colleagues. Networks are both simple in their symbols and rich in describing complexity. Second, networks are amenable to certain analytics that can help the architect compute relevant metrics.

Network approaches have been used in other settings such as inter-organizational networks and social networks. In those setting researchers have used several network metrics and provided interesting interpretations. One particular metric that is important in analyzing architectures is centrality.

Degree centrality of a node refers to the number of direct connections that a node has at a given point in time. A node with more direct connections to other nodes is considered to be more central than other nodes. For example, in Figure 1, node g has a degree of four and node h as a degree of 3.

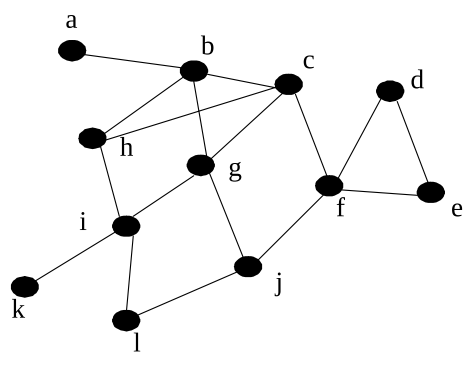


Figure 1: A sample network

Other centrality measures include the effect of indirect ties between nodes. (Node h may be directly connected to node b, but node h is also indirectly connected to node a (by going through node b)). For example, closeness centrality (one of the simpler measures incorporating indirect ties) measures the distance, using shortest paths, from each node to all other nodes. The node that can reach all other nodes in the fewest steps is considered most central.

Researchers and practitioners alike have been interested in the pattern of connections between nodes. Historically, researchers have assumed that the nodes are connected to one another randomly. As a result, if we were to plot the probability distribution of the number of links in the network, we expect to see a bell shaped curve that is centered on a mean number of links. However, in reality, when researchers observed connections between nodes in real networks they found that the distributions did not match the random expectation. In fact, they encountered several networks with a probability distribution that resembled what they called power-law distributions. This meant that these networks had a great majority of nodes with very few connections and few nodes with a great many connections. This result does not occur by accident. In fact, Albert-Laszlo Barabasi listed two conditions for making any network scale-free (Barabasi, 2002). First, the network should be growing with new nodes appearing at random times. Second, a new node should preferentially attach to an existing node based on the number of connections that the existing node already has.

Centrality and the way links are formed over time has implications for control and risk. When nodes occupy a central position they have the ability to influence the entire network. As a result of this, they gain importance. Within the architectural context, such nodes have to be managed closely or they could adversely impact the network as a whole. If nodes connect to other nodes randomly, complexity is not managed and the network represents a random network. If nodes connect to other nodes under the preferential attachment logic, a few nodes become very central – critical – to the network as a whole. These nodes may be very hard to modify, and could expose the firm to undo risk if they are not controlled or are uncontrollable. If nodes most often connect to nodes within their local domain, but have some links to other clusters of nodes, then a Small World network may emerge. These networks are more robust to perturbations than random networks. Researchers have created a metric called small world quotient, or Q, to measure this (Uzzi & Spiro, 2005). The measure of design preservation is captured through the network's Small World Quotient. A high value represents a high ratio of clustering to short path lengths, relative to a random network with the same node count and arc density.

Within the architecture domain, a few requirements stand out. Most enterprise architects are interested in identifying what they call the shared core. These are the set of components (systems, services, applications) that are used by most other components. Visualization packages such as Pajek, a freeware program (Batagelj & Mrvar, 2004), can be used to draw network diagrams and generate meaningful and replicable visualizations of very large networks (networks having hundreds of thousands of vertices). From within the many network drawing algorithms within Pajek, two algorithms – Fruchterman-Reingold (FR) (Fruchterman & Reingold, 1991) and Kamada-Kawai (KK) (Kamada & Kawai, 1989) – can be used to help the analyst identify the shared core. These algorithms use "spring-embedded" techniques to derive minimum energy states of the underlying networks.

A second requirement is identifying components that, if not working, will cause the most significant negative impact on the enterprise's ability to provide its services. Using an appropriate network model, the single most important node can be identified through an appropriate centrality measure. However, architects interested in risk management may be interested in the group of systems that represent those systems that must provide 24x7 availability or be maintained collectively to minimize disruption to the network. Identifying the critical set of systems is a greater challenge than identifying a single node. In order to do this, we utilize KeyPlayer (Borgatti & Dreyfus, 2005). It implements the KPP-NEG test (Borgatti, 2003) in which we identify the N nodes that have the potential to maximally disrupt the network.

The third requirement is similar to the second. Impact analysis helps us identify the set of systems that have the greatest impact on the rest of the network. Where traditional approaches would identify the direct and indirect systems affected by a proposed change to a specific set of systems, the impact analysis we suggest in this chapter is identifying the set of nodes that would have the largest impact on the network. In effect, this impact analysis finds the points of greatest leverage on the rest of the system. In order to do this, we again utilize Key Player but utilize the KPP-POS metric in which we identify the N nodes that collectively have the greatest reach in the network (Borgatti, 2003; Borgatti & Dreyfus, 2005). Although none of these nodes may be the most central, they may be the most important when we consider only their collective positional value in the network.

The final requirement (at least in this chapter), is to identify the best decomposition of a set of components that minimizes dependencies across clusters while maximizing dependency within a cluster. Enterprise architects are familiar with the notion of tight and lose coupling. Our challenge is to use network analysis techniques to create and analyze useful clusters. In our case study with FinServ (see next section), we have not yet done this analysis. Since FinServ has decided to adopt a service-oriented architecture (SOA) to rearchitect its applications and components as a set of services, this will be an important requirement. The problem of finding a grouping of vertices, such that for all vertices in each group they are powerfully connected within (homogeneity) and weakly connected to others (separation), remains an NP-complete problem since Karp (Karp, 1972) identified it as such. Social network analysis methods have provided polynomial-like algorithms, mostly applicable to boolean graphs (e.g., LS sets: Lawler (1973), Lambda sets: (Freeman, Borgatti, & White, 1991)), that have been implemented in various software packages (e.g., Ucinet)[1].

[1] An introduction to clustering methods can be found in (Everitt, Landau, & Leese, 2001).

A case study

In order to highlight the suggestions we have made above, we present the following case study to explore data collection requirements and interesting questions that network models of the enterprise architecture can help provide answers to.

Our research setting was within a financial services company (FinServ) that provides processing services to the investment management industry. In addition to our own primary data collection, we used details listed in (Westerman & Walpole, 2005) to inform our case study. FinServ has over a trillion dollars in assets under management. They provide full service transfer agency and accounting services globally.

Over the last couple of decades, FinServ has grown rapidly primarily through mergers and acquisitions and also by launching new services and entering new regions. As companies were acquired, each line-of-business (LOB) has maintained much of the decision rights to control most aspects of their (line-of) business, including design, sales, back-end processing and IT services. The resulting implied enterprise architecture is one with duplication of functionality, limited integration and applications with different look and feel. This was sustainable during the economic growth phase in the 90's. Each LOB was empowered to focus on and independently respond to specific opportunities and needs.

A combination of the dot-com crash and the associated stock marked decline coupled with increased competition caused FinServ to find that the market conditions quickly changed. This new market is very competitive. A commoditization of services has increasingly squeezed the company's profit margins. New demands on flexibility, agility, and improved time-to-market have become differentiators. Furthermore, as a result of 9-11, oversight bodies have stepped up the requirements on the financial industry with governance and compliance mandates (such as Sarbanes Oxley). These factors together have moved FinServ from being in a high growth profitable market to one where cost-cutting, activity-streamlining efficiency improvements are a necessity to stay competitive. While the strategic, long-term direction was clear, FinServ's existing architecture was not compatible with these changes. The attributes of the FinServ architecture had several challenges, such as tight coupling between systems and monolithic solutions, often not well documented, that duplicated functionality in multiple systems. This prevented FinServ from quickly responding to changing market conditions.

In response to this predicament, FinServ hired a new CIO and created a new corporate initiative to address the problems. The IT organization was changed, creating an enterprise architecture team, a management oversight committee, and an organization to provide company-wide shared application services. A FinServ enterprise architecture strategy was created and a Global Platform SOA-based enterprise architecture was defined. Their core mandate was to reduce IT spending though the elimination of redundant systems, componentization and exposure of core functionality through accessible services, redesign of strategic shared applications and components and the provision of mutually consistent customer access channels (web, voice, wireless, etc.). This organization invests in a strong enterprise architecture organization and actively practices architecture-driven application development with accompanying architecture governance via an Architecture Review Board. This involves investing in technology, processes, people, and organizational structures that can help them deliver on their mandate.

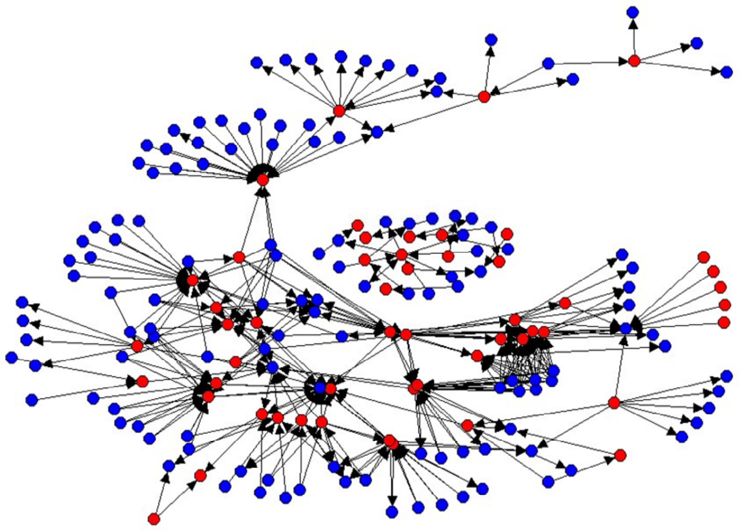
This multi-year effort required significant changes to FinServ's IT applications. To support the CIO's call for action, corporate IT organized an all-day planning session that resulted in the identification of the following goals:

1. Rationalize the application portfolio and eliminate redundant systems and applications
2. Restructure the legacy applications to create core components and re-architect highly reusable capabilities into shared services
3. Componentize and service-enable the components (create open interfaces to legacy code)
4. Create an enterprise architecture - a single platform across all LOBs
5. Enable significantly improved ability to design new customer-facing applications (agility)
6. Enable efficient integration of existing and new components.

We began our project by taking inventory of the applications within FinServ. The chief enterprise architect at FinServ had already collected information on applications and their dependencies. As a start, he collected dependencies that included data sharing, function calls and message passing. This information was stored in an Access™ database and updated constantly as new information trickled in from business units. Besides tracking systems and dependencies, the Access application generated Pajek and Key Player input files to support the visualizations and analysis.

Based on the inventory of all applications from FinServ, we drew the network diagram of the deployment architecture shown in Figure 2. Each node in the figure is a system within FinServ and each link represents a dependency described above.

Figure 2: Enterprise architecture



Node colors denote if the system is internal to FinServ or an external system that is used by FinServ. A blue color (darker shade) denotes an external system and a red color (lighter shade) denotes an internal one. Almost immediately one can see that several blue nodes occupy key positions within FinServ. This presents a challenge since they are both vital to the performance of other core systems and outside FinServ's control.

When we presented this picture to the architects it stimulated a lot of discussion. They observed that some aspects of the figure were obvious to them. For example, some red notes occupying the middle of the picture were their key transaction processing systems. What surprised them was the presence of blue nodes in the middle. This got them discussing these systems in great depth, not from a cost or performance perspective but from the ownership perspective. Are all the blue colored nodes truly outside FinServ's control? In other words, this was a data quality question. If they are indeed outside FinServ's control, why are these systems outsourced? What kinds of service level agreements do they have with those vendors? What are some of the risks? What platforms are these applications running on (UNIX, Linux, Windows, etc.)? What is the number of developers or users associated with each application? How far along (in terms of completion) are the projects associated with each application?

Another byproduct of the network analysis was the identification of the key players – the architectural control points. The top set of core nodes that were identified by our key player algorithm contained most of the core systems that the stakeholders had already identified. Given that they had close to 200 systems in their inventory, they were a little surprised about a few systems that showed up very high on our key player set. Upon further analysis, they were able to justify why these systems were important. A similar concern was raised about nodes containing either only incoming links or outgoing links. Once again, the question about data quality came up. An architect in the meeting raised an interesting point that he could use these pictures to determine the degree of difficulty in making changes to a system. He opined that systems with a large number of links, if these links were of different types, would be very expensive to modify. While there were many other smaller questions that came up, the chief architect summed up the session to us by saying that these figures had changed the nature of discussion from raw counts to business related issues and complexity/change management concerns.

Another issue that came up during the discussions was the presence of small clusters that were disconnected from the larger group. While on the surface these islands were disconcerting, some of the architects raised the possibility that there was missing information about the links. In fact, this session led them to look closely at the dependencies.

The visualizations were also valuable as a basis for communication among the architects, development managers, business managers, and senior management. The challenge for any technical group is explaining what they do and why things are as difficult as they are. For a perceived overhead function such as the enterprise architecture group, the challenge is greater because they also have to continuously justify their existence. It appears that the visualizations accomplish both tasks.

The graphics are simple enough in their nomenclature that people not familiar with the terminology can still understand them. To both senior management and line-of-business management, the visualizations provide an abstract view of the organization and its information flows. Managers can see how and where their systems fit into the larger picture. Managers can visualize the complexity and understand why certain changes are being recommended. The nodes represent the systems the managers' people use every day. The links highlight the managers' interdependencies with everyone else.

Initially we have limited the architectural view to a deployment view. The graphics represent the state of the enterprise systems and their dependencies as of today. By creating renderings of the information system at different points in time, people can see how the system evolves. In our prototype project, we maintained an Access database with an inventory of systems and dependencies. In time, this can be replaced by an asset management application dedicated to this task. Central to this type of architectural decision making is a running inventory of the organization's information system assets. Unfortunately, the missing piece in many asset management systems is the management of the dependencies. These aren't assets or liabilities in the traditional sense, yet are as important, if not more important, than the assets and liabilities usually tracked.

Creation of network models representing other architectural and/or organizational views can provide further significant insight of the current state of the architecture.

Data collection requirements and limitations

Based on our initial analysis, we presented our findings to the enterprise architecture group at FinServ. During our presentation, members of the architecture group had several clarification questions and also identified many new questions. Since they did not have enough time to discuss these new questions during that session, they decided to meet separately and have a separate brainstorming session. Coming out of the session were a set of broader questions and accompanying data collection requirements. Although our analysis was limited to an initial set of data gathered, we have had several rounds of follow-on discussions with stakeholders and as a result identified broader needs.

Enterprise architects planning to use our approach need to decide which architectural views are important to them. The architectural view is dependent on the question that the architect chooses to address. After the creation of the specific network model representation (the semantic meaning of arcs, nodes, colors, directional arcs, etc.) the necessary data has to be collected, such as, for example, information on systems, projects, the project team, and business users. Our technique hinges on finding the proper level of information. The best place to start is to consider a deployment architectural view of the set of existing components and their inter-relationships. On this project, we were able to begin discussions and interpretations based on a sample of the total data on systems. This was helpful because follow-on discussions resulted in the identification of new and relevant data elements.

Components can be anything from hardware, operating system and communication network components (e.g. Intel/Linux), application software components (e.g. customer service applications), business service components (e.g., shareholder services) and their respective interfaces, to software technology services (e.g., BPM, Security). For an architectural view of application software components, the modules of Oracle Financials, for example, constitute a single component. For each component, we need to know the components it calls, the components that call it, and information about the type of system it is (database, workflow, analytics, decision support, etc.).

For each component (piece of major software, hardware) in the network, other useful information to collect includes: the date it went into service, the project name associated with its introduction, prerequisite hardware and software platform information, and the names of the other systems it integrates with. For each integration point, we suggest collecting the information regarding the type of data exchanged and the method of exchange (e.g., CORBA, RMI, WebServices, RPC, synchronous or asynchronous messaging, batch file exchange, file system access, database access). The format of the data exchange should also be tracked. In addition to the static information we have just described, we recommend maintaining information regarding the frequency and volume of data exchange. This information can help the analyst better understand the nature of the dependency between the components within the production architecture.

Because the information system grows through projects, and projects are frequently the item that is budgeted, it is important to have each project name, description, components touched, project justification, and budget. In addition, data on the estimated time (elapsed and person-hours) and actual time (elapsed and person-hours), plus any extenuating circumstances, is also important. This data can be used to understand how interdependency between systems can affect project outcomes.

We also recommend collecting data regarding the people who work on the projects. Collect data on who worked on each project, who the project lead was, the role each person played on the project, and the components each person managed. This data can be used to develop insight into the social network of project teams, and aid in understanding how this network influences project outcomes.

On the user side, we recommend collecting data on which business users (groups) use which components of the system and which groups coordinate with which other groups independent of their use of the system. That is, it is useful to understand which groups work together, how they work together, and why they do so. With information on projects and business unit success, the analyst can start connecting the information system to project and business value. Based on our discussions with stakeholders, they consider the following to be successful outcomes: project estimation accuracy, degree of reuse, quality of the system, and changeability or flexibility, and ROI.

Conclusions

Viewing architecture from a network perspective provides two clear benefits. Networks provide a useful metaphor to communicate and interpret architecture and, based on their graph-theoretic underpinnings, provide a basis for analytics to compute metrics that may help explain variation in the performance of architectural and organizational flexibility, robustness, and risk management.

Our case study at FinServ helped create a decision support system and a set of metrics that are useful for making architectural decisions. In building the decision support system, we were able to identify the methodology and data required to successfully manage architecture based decisions. While the current version of the system requires users to input the information, in future versions FinServ is planning on automating the data collection and looking at live architectures.

Reference:

Akrich, M. (1992). The De-scription of Technical Artifacts. In W. E. Bijker & J. Law (Eds.), (pp. 205–224). Cambridge, MA: MIT Press.

Alexander, C. (1964). *Notes on the Synthesis of Form*. Cambridge: Harvard University Press.

Allen, T. J. (1977). Managing the Flow of Technology. In. Cambridge (MA): MIT Press.

Banker, R. D., Datar, S. M., Kemerer, C. F., & Zweig, D. (1993). Software complexity and maintenance costs. *Association for Computing Machinery. Communications of the ACM, 36*(11), 81.

Barabasi, A.-L. (2002). *Linked: The New Science of Networks*. Cambridge, MA: Perseus Publishing.

Batagelj, V., & Mrvar, A. (2004). Pajek Program for Large Network Analysis. 2004, from http://vlado.fmf.uni-lj.si/pub/networks/pajek/

Borgatti, S. P. (2003). The Key Player Problem. In R. L. Breiger, K. M. Carley, P. Pattison & N. R. C. U. S. C. o. H. Factors (Eds.), *Dynamic Social Network Modeling and Analysis: workshop summary and papers*. Washington, D.C.: National Academy of Sciences Press.

Borgatti, S. P., & Dreyfus, D. (2005). Key Player (Version 2.0). Harvard, MA.

Byrd, T., & Turner, D. (2000). Measuring the Flexibility of Information Technology Infrastructure: Exploratory Analysis of a Construct. *Journal of Management Information Systems, 17*(1), 167-208.

Chidamber, S., & Kemerer, C. (1998). A metrics suite for object oriented desing. *IEEE Transactions on Software Engineering, 20*(6), 476-493.

D'Souza, D., & Willis, A. (1999). *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley.

Dreyfus, D., & Iyer, B. (2006). *Enterprise Architecture: A Social Network Perspective.* Paper presented at the Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS-39), Koloa, Kauai HI.

Duncan, N. (1995). Capturing Flexibility of Information Technology Infrastructure: A Study of Resource Characteristics and their Measure. *Journal of Management Information Systems, 12*(2), 37--57.

Everitt, B. S., Landau, S., & Leese, M. (2001). *Cluster Analysis*: Arnold Publishers.

Freeman, L., Borgatti, S., & White, D. (1991). Centrality in valued graphs: A measure of betweenness based on network flow. *Social Networks, 13*(2), 141-154.

Fruchterman, T., & Reingold, E. (1991). Graph Drawing by Force-Directed Relacement. *Software Practice and Experience, 21*, 1129-1164.

Gasser, L. (1986). The Integration of Computing and Routine Work. *ACM Transactions on Office Information Systems, 4*(3), 205-225.

Glaeser, E., Kallal, H., Scheinkman, J. A., & Shleifer, A. A. (1992). Growth in Cities. *Journal of Political Economy, 100*(61), 1126-1152.

Gunter, C. A. (2000). Abstracting dependencies between software configuration items. *ACM Trans. Softw. Eng. Methodol., 9*(1), 94-131.

Henderson, J. C., & Venkatraman, N. (1993). Strategic alignment: Leveraging information technology for transforming organizations. *IBM Systems Journal, 32*(1), 4.

Iyer, B., & Gottlieb, R. M. (2004). The Four-Domain Architecture: An approach to support enterprise architecture design., *IBM Systems Journal* (Vol. 43, pp. 587-597): IBM Corporation/IBM Journals.

Kamada, T., & Kawai, S. (1989). An Algorithm for Drawing General Undirected Graphs. *Information Processing Letters, 31*, 7-15.

Karp, R. M. (1972). Reducibility among combinatorial problems. In M. a. Thatcher (Ed.), *Complexity of Computer Computations* (pp. 85-103): Plenum Press.

Kayworth, T., Chatterjee, D., & Sambamurthy, V. (2001). Theoretical Justifcation for IT Infrastructure Investments. *Information Resources Management Journal, 14*(3), 5--14.

MacCormack, A., Rusnak, J., & Baldwin, C. (2005). *Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code*.Unpublished manuscript.

Malone, T. W., & Crowston, K. (1994). The Interdisciplinary Study of Coordination. *ACM Computing Surveys, 26*(1), 87-119.

McKay, D., & Brockway, D. (1989). Building IT Infrastructure for the 1990s. *Stage by Stage, 9*(3), 1--11.

Messerschmitt, D., & Szyperski, C. (2003). *Software Ecosystems: Understanding an Indispensable Technology and Industry*. Cambridge, MA: The MIT Press.

Morris, C., & Ferguson, C. (1993). How Architecture Wins Technology Wars. *Harvard Business Review, 71*(2), 86-97.

Nezlek, G. S., Jain, H. K., & Nazareth, D. L. (1999). An integrated approach to enterprise computing architectures. *Communications of the ACM, 42*(11), 82-90.

Orlikowski, W. J. (2000). Using technology and constituting structures: A practice lens for studying technology in organizations. *Organization Science, 11*(4), 404-428.

Parnas, D. L. (1972). On the Criteria To Be Used in Decomposing Systems Into Modules. *Communications of the ACM, 15*(12), 1053-1058.

Pinch, T. J., & Bijker, W. E. (1987). The Social Construction of Facts and Artifacts. In T. Pinch (Ed.), *The Social Construction of Technological Systems* (pp. 17-50). Cambridge, MA: The MIT Press.

Richardson, G., Jackson, B., & Dickson, G. (1990). A Principle-Based Enterprise Architecture: Lessons from Texaco and Star Enterprise. *MIS Quarterly, 14*(4), 385-403.

Ross, J. (2003). Creating a Strategic IT Architecture Competency: Learning in Stages. *MIS Quarterly Executive, 2*(1), 31--43.

Sauer, C., & Willcocks, L. P. (2002). The Evolution of the Organizational Architect. *Sloan Management Review*(Spring), 41-49.

Thompson, J. (1967). *Organizations in Action*. New York: McGraw-Hill.

Uzzi, B., & Spiro, J. (2005). Collaboration and Creativity: The Small World Problem *American Journal of Sociology, 111*(2), 447-504.

Venkatraman, V. N., Lee, C.-H., & Iyer, B. (2005). Is Ambidexterity a Valuable Organizational Capability? An Empirical Test in Software Product Introductions, 1991-2002: Boston University.

Weill, P., & Broadbent, M. (1998). *Leveraging the New Infrastructure: How Market Leaders Capitalize on Information Technology*. Boston: Harvard Business School Press.

Westerman, G., & Walpole, R. (2005). PFPC: Building an IT Risk Management Competency [Electronic Version]. *CISR Working Paper*.

Wilson, M., & Howcroft, D. (2002). Re-conceptualising failure: Social shaping meets IS research. *European Journal of Information Systems, 11*(4), 236.

Yourdon, E., & Constantine, L. (1986). *Structured Design : Fundamentals of a Discipline of Computer Program and Systems Design*. Englewood Cliffs, NJ: Yourdon Press.

Zachman, J. A. (1987). A framework for information systems architecture. *IBM Systems Journal, 26*(3), 276--292.

Zhao, L., & Siau, K. (2002). Component-based Development using UML. *Communications of the Association for Information Systems, 9*(12), 207-222.