GUIDING NETWORK-CENTRIC
ARCHITECTURAL DESIGN: A STYLE-BASED APPROACH

Amine Chigani

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science

In

Computer Science and Applications

Advisory Committee:

James D. Arthur (Chair)

Shawn Bohner        Stephen Edwards

December 12, 2007

Blacksburg, VA

Keywords: Network-centric, software architecture, architectural style, style guide, architectural design, system of systems, viewtypes, component and connector

Copyright 2007, Amine Chigani
GUIDING NETWORK-CENTRIC ARCHITECTURAL DESIGN: A STYLE-BASED APPROACH

Amine Chigani
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
achigani@vt.edu

ABSTRACT

With the advance of reliable network technology, software development has progressed from traditional, platform-centric software construction to network-centric software evolution. An evidence of this change is largely reflected in the technologies that are supporting the emerging theory of Network-Centric Operations (NCO). Amongst these technologies is software architecture as a software engineering sub-discipline. Although the concepts of network centricity are widely recognized within the software and system engineering communities, no unified characterization of network-centric software systems is unanimously adopted. The state-of-the-practice is characterized by differing interpretations about how we should design and implement this class of systems. In this research, our focus is twofold: 1) Providing a characterization framework to reason about network-centric software systems and 2) introducing one solution approach to designing this class of system based on a new architectural style, the network-centric architectural style. In so doing, we set the stage for the software architecture community to analyze the “fitness of use” of current architectural styles and architecture design practices within this new network-centric paradigm. In addition, we set the stage for our continued research that will address further software engineering challenges pertinent to network-centric software systems, which include capability-based requirements engineering and quality attributes-based design.
DEDICATION

I dedicate this work to my beautiful wife, Jihane Najdi. Her support was and is still beyond that of a wife. She has been involved in every aspect of my graduate career decisions and is still the biggest supporter of my choice to pursue a PhD degree. She has supported me emotionally and financially throughout this journey. For that, I dedicate this work to her.

I also dedicate this work to my family, especially my parents, Sadik and Zahra, whom I hope I have made proud today. They have always pushed me to work hard and succeed in whatever career choice I made. I love you mom and dad. I would also thank all my siblings and their families for all their support: Khadija, Fatima, Mohammed, Mostafa, Touria, Amina, and Yusuf.
ACKNOWLEDGEMENTS

I would like to begin by expressing my thanks, gratitude, and praise to Allah (Almighty God) for His help, guidance, and mercy that He has showered upon me throughout this journey. Without His permission and help, this work would not have been accomplished. Alhamdu Lilah (All thanks are due to Allah).

I would also like to express my deepest gratitude to my academic and research advisor, Dr. Sean Arthur. His guidance and support – academic, professional, and personal – was the main reason that this thesis work started and is now complete. For this, I thank him and acknowledge his effort and dedication to make me a better student, a better researcher, and a better person. I hope that one day I will be able to provide the same level of personal and professional support to my students.

In addition, I owe special thanks to Dr. Bohner who had a direct impact on the direction and success of my research and the writing of this thesis. I feel fortunate that I had the chance to collaborate and conduct research with him. I also thank both Dr. Edwards and Dr. Cameron for their involvement in this work.

Before I end, I would like to thank Ramya Ravichandar who was a good friend and good research colleague. Her support and advice during the times when things were not going well was helpful.

Special thanks go to the Muslim Student Association at Virginia Tech and the Muslim community in Blacksburg whose members provided me with the spiritual atmosphere that allowed me to grow and prosper as a person, and to succeed as a graduate student.

Again, I thank Allah for all his bounties: The bounty of having a great advisor, the bounty of working with a caring committee, and the bounty of having a supporting community and friends.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
<tr>
<td>1. RESEARCH OVERVIEW</td>
<td>1</td>
</tr>
<tr>
<td>1.1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.2. SYMPTOMS OF THE PROBLEM</td>
<td>2</td>
</tr>
<tr>
<td>1.3. PROBLEM STATEMENT</td>
<td>4</td>
</tr>
<tr>
<td>1.4. ISSUES TO DEFINE A SOLUTION</td>
<td>4</td>
</tr>
<tr>
<td>1.5. OUR SOLUTION APPROACH</td>
<td>6</td>
</tr>
<tr>
<td>1.5.1. characteristics of Network-centric software systems</td>
<td>7</td>
</tr>
<tr>
<td>1.5.2. A framework within which we can define a solution</td>
<td>8</td>
</tr>
<tr>
<td>1.6. THE NETWORK CENTRIC ARCHITECTURAL STYLE</td>
<td>9</td>
</tr>
<tr>
<td>1.6.1. Network-Centric Architectural Style Guide</td>
<td>9</td>
</tr>
<tr>
<td>1.6.2. summary of the Network-Centric Architectural Style</td>
<td>11</td>
</tr>
<tr>
<td>1.7. BLUEPRINT OF THE THESIS</td>
<td>11</td>
</tr>
<tr>
<td>2. MOTIVATION AND PROBLEM EVOLUTION</td>
<td>12</td>
</tr>
<tr>
<td>2.1. MOTIVATION</td>
<td>12</td>
</tr>
<tr>
<td>2.2. BACKGROUND ON NETWORK-CENTRIC SYSTEMS AND ARCHITECTURES</td>
<td>14</td>
</tr>
<tr>
<td>2.2.1 Network-Centricity in a Military Context</td>
<td>15</td>
</tr>
<tr>
<td>2.2.2. Reconciling the Terms: Network-Centric and Software Systems</td>
<td>15</td>
</tr>
<tr>
<td>2.2.3. Network-Centricity in a Research Context</td>
<td>16</td>
</tr>
<tr>
<td>2.2.4. The Current State-of-the-Practice</td>
<td>17</td>
</tr>
<tr>
<td>2.2. DOMAIN AND ORGANIZATIONAL-SPECIFIC NETWORK-CENTRIC INITIATIVES</td>
<td>18</td>
</tr>
<tr>
<td>2.2.1. Raytheon’s Network-Centric Systems</td>
<td>19</td>
</tr>
<tr>
<td>2.2.2. NASA’s Integrating Systems for Joint Operations</td>
<td>20</td>
</tr>
<tr>
<td>2.2.3. Boeing’s integrated Battlespace Vision</td>
<td>21</td>
</tr>
<tr>
<td>2.2.4. Summary</td>
<td>22</td>
</tr>
<tr>
<td>2.3. INITIATIVES HAVING NETWORK-CENTRIC FOCUS</td>
<td>23</td>
</tr>
</tbody>
</table>

[v]
### 3. NETWORK-CENTRIC SOFTWARE SYSTEMS: DEFINITION, CHARACTERISTICS, AND CHALLENGES

#### 3.1. Definition of Network-Centric Software Systems

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1. Discussion</td>
<td>36</td>
</tr>
<tr>
<td>3.1.2. Contextualizing Network-Centric Software Systems</td>
<td>39</td>
</tr>
<tr>
<td>3.1.2.1. Military Context</td>
<td>39</td>
</tr>
<tr>
<td>3.1.2.2. Industry Context</td>
<td>41</td>
</tr>
<tr>
<td>3.1.2.3. Scientific Research Context</td>
<td>41</td>
</tr>
</tbody>
</table>

#### 3.2. Key Characteristics of Network-Centric Software Systems

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1. Structural Characteristics (Static)</td>
<td>43</td>
</tr>
<tr>
<td>3.2.1.1. System of Systems Perspective</td>
<td>43</td>
</tr>
<tr>
<td>3.2.1.2. Network-Based Configuration</td>
<td>44</td>
</tr>
<tr>
<td>3.2.2. Operational Characteristics (Dynamic)</td>
<td>46</td>
</tr>
<tr>
<td>3.2.2.1. Emergent Behavior</td>
<td>46</td>
</tr>
<tr>
<td>3.2.2.2. Decentralized Control</td>
<td>47</td>
</tr>
</tbody>
</table>

#### 3.3. Challenges to Designing Network-Centric Software Systems

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.1. Standardization</td>
<td>48</td>
</tr>
<tr>
<td>3.3.2. Scalability</td>
<td>49</td>
</tr>
<tr>
<td>3.3.3. On-Demand Composition</td>
<td>49</td>
</tr>
<tr>
<td>3.3.4. Robust Connectivity</td>
<td>50</td>
</tr>
<tr>
<td>3.3.5. Security and Trust</td>
<td>51</td>
</tr>
<tr>
<td>3.3.6. Test and Evaluation</td>
<td>52</td>
</tr>
</tbody>
</table>

#### 3.4. Summary and Transition to Chapter IV

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4. Summary and Transition to Chapter IV</td>
<td>53</td>
</tr>
</tbody>
</table>

### 4. NETWORK-CENTRIC ARCHITECTURAL STYLE GUIDE

#### 4.1. Definition of the Network-Centric Software Systems – Revisited

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1. Definition of the Network-Centric Software Systems – Revisited</td>
<td>55</td>
</tr>
</tbody>
</table>

#### 4.2. Motivation for a Style-Based Approach

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2. Motivation for a Style-Based Approach</td>
<td>56</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WAAS Scalable Architecture</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>Oracle’s Network-Centric Reference Architecture</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>NCOIC Technical Deliverables</td>
<td>26</td>
</tr>
<tr>
<td>4</td>
<td>SWIM: An Information Management Architecture</td>
<td>27</td>
</tr>
<tr>
<td>5</td>
<td>Geon System Architecture</td>
<td>34</td>
</tr>
<tr>
<td>6</td>
<td>Key Characteristics of a Network-Centric Software System</td>
<td>37</td>
</tr>
<tr>
<td>7</td>
<td>Aspects of Developing Network-Centric Software Systems</td>
<td>57</td>
</tr>
<tr>
<td>8</td>
<td>Depiction of a Basic Tenet of Network-Centric Software Systems – Network-Based Interaction</td>
<td>62</td>
</tr>
<tr>
<td>9</td>
<td>A Summarized Style Guide of the Network-Centric Architectural Style</td>
<td>63</td>
</tr>
<tr>
<td>10</td>
<td>A Sample Configuration to Carry Out a Network-Centric Task</td>
<td>65</td>
</tr>
<tr>
<td>11</td>
<td>A View of Heterogenous Nodes Communicating Via Links</td>
<td>73</td>
</tr>
<tr>
<td>12</td>
<td>Steps of Our Solution Approach</td>
<td>77</td>
</tr>
<tr>
<td>13</td>
<td>Potential Areas of Continued Research Work</td>
<td>81</td>
</tr>
<tr>
<td>14</td>
<td>Capabilities as an Input to Architectural Design</td>
<td>82</td>
</tr>
</tbody>
</table>
LIST OF TABLES

TABLE 1. A SUMMARY OF THE NETWORK-CENTRIC ARCHITECTURAL STYLE GUIDE 10
TABLE 2. NETWORK CENTRICITY FOCUS AREAS OF VARIOUS DOMAINS 28
TABLE 3. AN ABREVIATED STYLE GUIDE OF THE SOA ARCHITECTURAL STYLE 61
TABLE 4. THE NETWORK-CENTRIC ARCHITECTURAL STYLE GUIDE 72
CHAPTER I

1. RESEARCH OVERVIEW

1.1. INTRODUCTION

The ubiquity of the network and the ability to deploy software over a network has given rise to network-centric software systems. As with many technology shifts, advancements in networking technologies have changed the underlying assumptions for producing effective system solutions. The “system of systems” perspective now dominates much of the engineering as new systems are composed of multiple interconnected systems to support emerging missions. Moreover, the timeline of change has shifted from build customized systems for optimum efficiency to integrate and augment systems for agile response to changing missions [40]. The cycle time and economic implications are substantial as systems can be fielded in shorter periods of time integrating existing systems.

As a consequence, software development efforts have seen an emergence of this new culture that focuses on the integration of existing and new software systems to tackle larger and complex computing problems. A fundamental aspect of this culture is its substantial reliance on networked communications among the different system elements [1]. One reason behind this shift is the need to reach beyond tightly-coupled environments to access data and functionality that reside on remote systems that run on different platforms, and which are possibly owned and managed by different organizations. Another reason is the dynamic and complex structures of today’s organizations, where the computing resources of an organization can span multiple national and international locations [2].

This phenomenon has become prevalent in both government and industry technology initiatives. On the one hand, the U.S. department of Defense has coined the term network-centric operations to reflect its goal to transform all its agencies into a network of collaborating agents that share data and applications in real time to support the missions of...
the warfighter. However, the ideals of network centricity would not have been adopted by DoD if it were not for the advances achieved by a number of commercial technologies. For instance, this culture shift is drastically influenced by the advent of revolutionary network technologies, fast processing power, increased security measures, short time-to-market development cycles, improved software engineering practices, and increased communication standards. The DoD saw in these advances the opportunity to implement its vision of global dominance in the area of warfare and national security.

On the other hand, there has been a paramount impact of information and network technologies on business organizations. This effect drives the changes we observe on the way business services are provided. Due to the interconnected nature of the today's software operating environments, organizations have recognized the need to move towards a development model that utilizes data and application integration instead of development from the ground up. This new development model is what is now known as network-centric computing.

In this thesis, we examine and discuss several aspects of network-centric software systems and address issues pertinent to their architectural design. We first identify the symptoms of the problem at hand, which is how we can characterize network-centric software systems. We then identify the operational and structural characteristics of network-centric software systems. Next, we closely examine perspectives from which this problem can be addressed. Finally, we employ architectural design principles to introduce and outline a new architectural style that helps realize the characteristics of network-centric software systems.

1.2. SYMPTOMS OF THE PROBLEM

In the preliminary research, we have focused on studying existing network centric architectures and frameworks. This effort has revealed diverse trends in the way these attempts try to achieve the goals of network centricity. We have observed several indications that there is a need for a broad and accepted understanding of how to design network-centric software systems.
While “network-centric” is a term coined by the Department of Defense [3], the concept has been applied in industry (Boeing, Lockheed-Martin, and others) with shaping systems for agile operations. We view the introduction of the network-centric operations (NCO) as the pinnacle of the shift into the network-centric software development model. One reason is that many DoD contractors and research partners are now involved in contracts and research grants that focus on developing and providing tools and capabilities for all the supporting technologies of NCO. The outcome of such initiatives has been network-centric systems and technologies that are targeted to support DoD mission. Unfortunately, the requirements for existing network-centric systems and architectures are ambiguous and do not necessarily conform to one characterization.

Additionally, commercial organizations also recognize the necessity to streamline their business processes [4] through the integration of their diverse computing resources. Sharing the same strategic goals of NCO – which is the use of information technology to achieve business goals more efficiently – many commercial organizations have developed network-centric frameworks and architectures for their computing infrastructures. These architectures and frameworks also do not conform to a unified formal characterization.

Although network-centricity as a term is widely recognized among software engineers as evidenced by the discussion above, we see different characterizations of network-centric systems that range across the spectrum. The main reason is that not only that network-centric computing is a new development paradigm but also is an intricate one. The software engineering community is still trying to identify the distinguishing characteristics of network-centric systems, the set of requirements these systems need to meet, and the best way to design this class of systems. Another related reason is the lack of an approach to mapping our understanding of network-centricity concepts into the activities of software design. This has resulted in architectural solutions that do not comprehensively address the characteristics of software systems that are intended to be network-centric since these characteristics have not been fully embellished.

One of the primary challenges to the development of network-centric software systems is interoperability, which is addressed by evolving and improving network architecture
frameworks. However, what we observe is that there is a missing linkage between these network architectures and the software engineering processes. Many believe that interoperability is only a challenge to the networking community and that by developing more robust network communication technologies the problem is solved. What we have seen is that these technologies are only enablers to achieve the full premise of network-centric computing. Without the establishment of sound software engineering processes that support the missions of network-centric software systems, we will not be able to address the challenges that network-centricity brings about.

The software engineering community has recognized the importance of network-centric computing and has engaged in the development of network-centric software; however, the issue is that we have reached a state where we see a need to step back and look at how we can bring unity to the way we reason about network-centric software systems.

1.3. PROBLEM STATEMENT

Recently, we have noticed the emergence of a new software development model focused on effectively facilitating the construction of network-centric software systems. However, no cohesive approach to designing this class of systems has been identified. As a result, different organizations have adopted their own visions of what a network-centric system is. These visions are dissimilar and therefore the resulting systems do not exhibit similar characteristics. Thus, we infer that a formal characterization of network-centric software systems from a software engineering perspective is necessary. We also conclude that there is a need for a cohesive approach to the design of this class of systems.

1.4. ISSUES TO DEFINE A SOLUTION

Before discussing our solution approach in any detail, we present several considerations that challenge defining a characterization of network-centric software systems.

Recently, the idea behind network centricity has become widespread and everyone is talking about it. However, each person (or organization) who is involved in network-centric computing views network centricity from a different perspective. Some view it from
the network architecture perspective, some look at it from the software engineering aspect, and others see it as the buzzword that can be used to market their products. The result of these various perspectives is a range of characterizations of how network-centric software systems should be developed that differ across the board. In addition, the software engineering community has yet to agree on a common set of characteristics that represent network-centric software systems.

Moreover, even with defined common characteristics of network-centric software systems, the software engineering process to develop these systems will be incomplete without a systematic synthesis of new and traditional development techniques that accommodate those distinct characteristics. These techniques must take into consideration all phases of the software engineering process including requirements, design, implementation, testing and integration. Our focus in this thesis focuses primarily on the design phase and in particular architectural design.

A related challenge to the absence of a unified definition of network-centric software systems is the existence of government and non-government initiatives that develop systems that are network-centric in nature. These systems may, in fact, be network-centric; yet, in most cases, they are not developed using the same approach. This makes it difficult to construct knowledge that can be shared and reused among the software engineering community and that can actually be taught in academic institutions to future software engineers. In situations like these, the challenge is to construe the right set of characteristics for network-centric software systems from existing systems in order to be able to construct development approaches that achieve those characteristics.

Another challenge to characterizing network-centric systems is to define where to start this characterization within the development life cycle. Do we start at the low-level design or at the implementation level? An insightful analysis of network-centric software systems is needed to determine where these characteristics can be best addressed. It might be that the whole software engineering process needs to be evaluated for “fitness of use” when it comes to network-centric computing. However, from our preliminary research, we conjecture that first consideration should be given to the architectural techniques that
support the development of network-centric software systems. The reason is that the components of a network-centric software system are often developed independently and thus the system emerges only through the interaction of the components. The architect must express an overall structure (architecture) for these systems largely through the specification of communication standards. Thus, to accurately address the development challenges that network-centric software systems lead to, additional architectural techniques must be employed.

Finally, from our review of the literature of network-centric computing, we observe a lack of a network-centric computing nomenclature. An example is the use of the terms network-centric framework and software architecture, which are used interchangeably in much of the literature. In fact, these terms refer to two different things. For instance, the .NET framework refers to an underlying implementation framework upon which we can build network-centric applications. On the other hand, the network-centric software architecture for an application built using the .NET framework refers to the design artifact that represents the overall structure of that application. There is a need to develop a common language that can be used to reason about network-centric technologies.

The abovementioned list of issues is substantiation as to why we view our work as a research challenge. Inherently, this list also drives our solution approach and it represents our motivation to investigate ways to address this problem.

1.5. OUR SOLUTION APPROACH

There are two major components to our solution approach. First, we identify the distinguishing characteristics of a network-centric software system from other complex systems. Within this step, we differentiate between the operational characteristics and the structural ones of network-centric software systems. The second component of our solution approach focuses on the design of network-centric software systems. We do so by identifying the architectural framework within which we can describe architectures for network-centric software systems. Then, we introduce the network-centric architectural style as a solution within this framework.
1.5.1. CHARACTERISTICS OF NETWORK-CENTRIC SOFTWARE SYSTEMS

Network-centric software systems are multifunctional systems that exhibit unique operational characteristics. In our preliminary research, we have identified few operational characteristics that distinguish network-centric systems from other systems. A network-centric software system has:

- An underlying networked configuration that embodies the runtime environment on which the system’s components interact and limits components’ interaction to information exchange
- An emergent, dynamic runtime behavior, which means that the system’s actual interacting components are not necessarily known until runtime and that the overall functionality of the system emerges from the collaborative behaviors of the components.
- A fluid, dynamically-defined control, which means that control over the system functionality is not necessarily owned by a particular component; rather, this control changes based on which function the system is performing and which component has initiated the system’s execution. This control can be either strategic or tactical control.

In addition to their operational characteristics, network-centric software systems have distinctive structural characteristics. We have identified several of these characteristics. Network-centric software systems are:

- Multi-functional systems that have a “system of systems” perspective. The components of these systems are in and of themselves large systems that work together towards the same goal.
- Generally loosely-coupled systems. However, we have seen a spectrum of how the constituent components of these systems are coupled. It ranges from very tightly-coupled components to loosely-coupled elements that interact chiefly at the interface level through data and information exchange.
Systems that have underlying networked architecture where data and applications are deployed on the network. Traditionally, advances in network technologies have allowed access to data in geographically remote data repositories. With network-centric computing, the focus is on providing efficient ways to access remotely-located applications.

In chapter III, we discuss both the operational and structural characteristics of network-centric software systems in great detail. We also detail why we identified these specific characteristics and present evidence from the literature.

1.5.2. A FRAMEWORK WITHIN WHICH WE CAN DEFINE A SOLUTION

The problem at hand is to identify a formal characterization of network-centric software systems and to find a way to effectively construct this class of systems. In our research, we focus on identifying an architectural solution that allows us to design network-centric software architectures for these systems. In order to achieve this goal, we have examined existing software engineering practices to identify frameworks within which we can define a solution. Building upon the work done by Clements et al [5], we conclude that this solution must be in terms of a new architectural style, added to the reservoir of existing styles, which will enable architects to design systems that answer to both the demands of network centricity and their respective problem domain. The reason is that the concept of a style defines the features of a family of software architectures for a particular class of systems [6, 7], and the network-centric software systems are one such class.

Generally, architectural design has long been accepted as an essential ingredient of a well-designed software system. During this phase, network-centric software systems can be perceived from different perspectives. Clements et al refer to these perspectives as viewtypes [5]. Relevant to our discussion here is the component-and-connector viewtype, C&C viewtype for short. This viewtype corresponds to the way architects look at the software system as a set of elements and their interactions at runtime. We consider the C&C viewtype a relevant way of describing and documenting an architectural style.
Based on the characteristics we have identified, we can view network-centric software systems based the following:

- The decomposition of their constituent sub-systems
- The emergent behavior of the sub-systems and the communication mechanisms among them at runtime
- The allocation of the sub-systems to their underlying environment

In order to be able to build software architectures for this class of systems, an architect will need to employ several architectural styles. In this research, we begin to define a new architectural style that combines the characteristics of several architectural styles in order to bring unity to the way network-centric software systems are constructed. These are the styles we intend to use:

- Decomposition style to show the boundaries of the sub-systems
- Peer to Peer and communicating processes styles to show the independence of the sub-systems and the concurrency of their functions
- The allocation style to show the distributed and heterogeneous nature of the system

Clements et al [5] has a detailed discussion about these styles. In the following section we present an outline of the network-centric architectural style. The format of documentation is based on the C&C viewtype structure used in [5].

1.6. THE NETWORK CENTRIC ARCHITECTURAL STYLE

An essential part of documenting a new style is to develop a style guide that records the specialization and constraints the style imposes on its elements and their interactions. The following is an overview of the network-centric architectural style.

1.6.1. NETWORK-CENTRIC ARCHITECTURAL STYLE GUIDE

Table 1 shows a summary of the network-centric architectural style.
### Elements

**Components type:** Independent, active software nodes that have data exchange and task invocation ports (interfaces)

**Connectors type:** Data exchange and task invocation links, which facilitates communication among the nodes.

### Relations

**Attachment** relation associates a node’s port with another node port. Port compatibility must be enforced.

### Computational Model

Nodes are configured dynamically to collaboratively carry out network-centric tasks. A task can be initiated by any node in the system.

### Communication Model

Nodes are connected by means of links between them. A node can communicate with multiple nodes via proper port-link combinations.

### Elements Properties

**Node:**

*Name:* should suggest its functionality

*Type:* defines general functionalities and services of the node

*Ports:* list the types of ports the node has

*Port Properties:* depend on the type of the network communication

**Link:**

*Name:* should suggest the nature of the interactions it supports

*Type:* Data exchange or task invocation link

*Parameters:* defines the nature of the interaction and the required parameters to carry out that interaction

*Other Properties:* depending on network communication type, it may include protocol of interaction and performance values

### Topology

There is no topological model to this style. Nodes connect and disconnect to other nodes at will. The system has a “changing” topology that is unrestricted and that is based on the task being carried out.

| TABLE 1. A SUMMARY OF THE NETWORK-CENTRIC ARCHITECTURAL STYLE GUIDE |
1.6.2. SUMMARY OF THE NETWORK-CENTRIC ARCHITECTURAL STYLE

This style consists mainly of nodes and links that facilitate communications among the nodes. The analogy of an application of network style is that of a dynamic data structure. When we define a dynamic data structure such as linked list for instance, we do not know what an instance of that data structure will look like at runtime. However, when we design this data structure, we can define how new nodes can be added and removed. Similarly, the network-centric style allows architects to identify the overall structure of how nodes work together, and what kind of links can be used to interact with one another independent of the participating nodes and links at runtime. The network-centric style also helps in moving critical design decisions to the level of architecture, which in turns makes it possible to address certain quality attributes and to perform risk analyses to avoid misallocation of resources.

1.7. BLUEPRINT OF THE THESIS

The remainder of this thesis is organized as follows. Chapter II discusses the background material and motivation behind this research. It also contains a literature review about the progression of the network-centric concepts, initiatives, and technologies. This review includes initiatives from industry, military, and academia. Chapter III proposes and substantiates our definition of what constitute a network-centric software system. It discusses the key operational and structural characteristics of network-network software systems, as well as the issues pertinent to the architectural design of this class of system. Chapter IV introduces several architectural concepts such as software architecture definition, architectural styles, and viewtypes. It also focuses on the components of our solution approach manifested in the network-centric architectural style guide and on how this style addresses the key operational and structural characteristics of network-centric software systems. In Chapter V, the last chapter, we summarize our research work and provide plans for future work.
CHAPTER II

2. MOTIVATION AND PROBLEM EVOLUTION

This section discusses the drivers and background of the research problem we are investigating. Primarily, we present our analysis of the implications of the recent paradigm shift in developing large-scale software systems. Secondly, we survey the literature to capture the evolution of the concepts that have led to the current state of understanding and developing network-centric software systems.

2.1. MOTIVATION

Network-centric computing is the culmination of the development shift that has been occurring over the past few years. The changing nature of missions and business goals require software systems to be increasingly adaptive and responsive to such changes. In addition, the complexity of the computing problems software developers face nowadays drives the need to reach beyond locally-available resources to geographically-remote data and application stores. Recognizing this need, software developers have adopted different approaches to develop network-based solutions that scale to the level of complexities for which software systems are deployed to do. For instance, development models such as distributed computing, grid computing, and recently service-oriented computing are outcomes of initiatives to achieve data and application integration. These models suggest several aspects of network-centricity, but not all.

Recently, researchers and practitioners have increasingly focused on reaping the benefits network-centricity can bring to today’s software development efforts. Consequently, a whole vocabulary has been developed to talk about network-centricity. Terms such as network-centric architecture, network-centric operations or warfare, and network-centric computing or engineering, and interoperability are widely used in the literature both in government and non-government publications. Yet, what do these terms mean? Better yet, what do they mean within each context in which they are used. Our review of the network-
centricity literature clearly shows inconsistencies and sometimes contradictions in the
definitions given to these terms. More importantly, talking to practitioners directly
involved in developing network-centric systems and technologies reinforces our finding
that the network-centric terminology used is still immature. For example, it is not
uncommon to find two practitioners within the same domain (military, navy, banking, IT
solutions, etc.) having entirely different understandings of network-centric systems and
architectures. We attribute these inconsistencies to the lack of a common vernacular that
can provide the grounds for fruitful discourse to reason about network-centricity.

The question that must be asked is “what is the cause of these inconsistencies, and what is
the impact they have on today’s network-centric solutions?” The answer to the first part of
the question is straightforward. The notion of network-centric computing is nascent and
still maturing. Yet, we believe there are enough first principles and background from which
we can begin to develop a coherent language to reason about network-centric software
systems. Conversely, to address the second part of the question, we need an insightful
analysis of the current network-centric initiatives and systems. We discuss the details of
this analysis in the following section of this chapter. Here, we provide the findings of our
investigation. The conclusion is that the disparate understandings of the concepts and goals
of network centricity has lead to the emergence of various development approaches for
network-centric software systems and their architectures. These approaches often have
different characterizations of what a network-centric software system is. One downside to
this is that it becomes difficult to construct knowledge that can be shared and reused
among the software engineering community. Another downside is that this knowledge
cannot actually be taught in academic institutions to future software engineers since it
have not been codified to be used in academic curricula.

Our preliminary research reveals that network-centric computing is at a point of
maturation that requires a holistic view of how this new model affects software
engineering practices. We consider a formal approach to characterizing network-centric
software systems as the beginning for a solution to this problem. Examining the current
state-of-the-practice, we conclude that such an approach is lacking. Therefore, we propose
an approach to characterizing network-centric software systems. This approach starts by
identifying the distinguishing characteristics of network-centric software systems, and then defining what it means for a software system to be network-centric. Within this approach, the focus is on the architectural design techniques required to accommodate the emerging requirements of network-centric software systems. We will discuss our solution approach in details later in this thesis.

So, how did the development culture change into a network-centric model of operations? In the following section, we give a brief background about the idea of network-centric computing and the origins of the concepts of network-centricity. We employ references from the literature to show the evolution of these concepts.

2.2. BACKGROUND ON NETWORK-CENTRIC SYSTEMS AND ARCHITECTURES

Recent years have revealed a clear transformation from a platform-centric to a network-centric software development model. Conventional software use focuses on applications installed and updated manually [3, 8] on local devices such as PCs, mainframes, and others. Increasingly, the need to use other interface devices to access remote data and computational resources has become inevitable. Additionally, technological breakthroughs in hardware and network communications have opened the door for the software engineering community to address larger and more complex problems that were, until more recently, unsolvable. Furthermore, software acquisition has raised numerous challenges to integrate existing applications with the acquired ones. As a result, distributed, grid, service oriented computing, and other similar disciplines have emerged as products of the shift towards network-centric computing.

“Network-centric” is used loosely in many areas of the software engineering including software architecture [8, 9]. Understanding the origin and background of this term enables us to use it more accurately and to describe what it means in the context of software development. The following subsections (2.1.2.1, 2.1.2.2, 2.1.2.3 and 2.1.2.4) detail how different sectors (military, industry, and research) adapt the concepts of network-centricity to accomplish their business and mission goals.
2.2.1 NETWORK-CENTRICITY IN A MILITARY CONTEXT

The term “network-centric” has gained a widespread use after the introduction of Department of Defense network-centric warfare (NCW), later called operations (NCO). The history of the term NCO can be traced as far back as 1996 when Admiral Williams Owens introduced the concept of “system of systems” in one of the papers published by the Institute for National Security Studies [10]. However, Admiral Arthur K. Cebrowski and John Gastska formally introduced the term NCW in 1998 in an article published in the Naval Institute Proceedings [11]. The concepts were later given depth in the book, Network Centric Warfare [12]. The book derives this new theory of warfare from a series of case studies on how businesses are using information and communication technologies to improve situation analysis, accurately control inventory and production, as well as monitor customer relations.

Likewise, NCO is an emerging theory of war that seeks to translate an information advantage into a competitive warfighting advantage through the robust networking of well-informed, geographically-dispersed forces allowing new forms of warfighting organizational behavior [12]. NCO’s basic tenets include:

- Utilizing technological advantages to support warfighters in the battlefield
- Networking all systems used by US armed forces
- Achieving shared awareness of the battlefield amongst all members of the US armed forces [13].

To achieve its goals, NCO depends on many technologies including network architectures, satellites, radio bandwidth, unmanned vehicles, nanotechnology, processing power, and, most importantly for this research, software systems.

2.2.2. RECONCILING THE TERMS: NETWORK-CENTRIC AND SOFTWARE SYSTEMS

Network-centricity is not just connectivity across systems or nodes, but between people, information, and cognitive domains. It stresses sharing information to achieve situational awareness, which leads to increased speed of decision-making within various operational
environments. Network centricity is technical, operational, and behavioral. It has been the driving force behind the ongoing transformation of government and commercial business operations.

Network-centric software systems focus substantially on their communication element. To accomplish the ideals of network-centricity, these systems must achieve effective application and data integration. This integration is achieved by taking various systems on different platforms (i.e., OSs), built with different implementation frameworks, expressed using different programming languages [9], accessing different remote and local data repositories, and integrating them into robust systems for supporting critical business processes, scientific research programs, and mission threads.

A key reason for becoming network-centric is to be able to assemble software systems by integrating a mix of existing and new applications, and ensuring that the end-product is capable of integrating with other net-ready applications. In general, this is the distinguishing characteristic between network-centric and distributed system. The latter is engineered mostly for performance improvement. The former is engineered to reuse existing resources and integrate them to form larger and more complex systems over the network. For network-centric software systems, having their elements geographically distributed is a constraint rather than a choice of the architect.

2.2.3. NETWORK-CENTRICITY IN A RESEARCH CONTEXT

Much of the literary work related to network-centric computing has been introduced within the context of military operations. However, several academic researchers have substantial contributions to evolving the field to where it is now. As early as the year 1998, Mary Shaw recognized the emergence of a new approach to software development enabled by the prevalent use of the Internet [14]. The premise of this new approach is computing through dynamically-formed, task-specific coalitions of distributed, autonomous resources. These coalitions are characterized by a lack of direct control over the independently developed and managed components that constitute them. In addition, David Garlan takes the concept of coalitions of resources a step further. In analyzing the challenges facing the
emerging field of software architecture, Garlan identifies several architectural design challenges related specifically to the new model of software development: Network-centric computing [8]. Different research initiatives have emerged that focus on each of the challenges Garlan identifies.

For instance, interoperability is one of these challenges. The focus of the Integration of Software-Intensive Systems (ISIS) initiative [15] conducted by the Software Engineering Institute (SEI) is addressing the gaps between the network-centric vision and the status and capabilities of technologies that are being targeted to fulfill this vision. This is certainly a daunting task. It requires that new methods and techniques be developed to assist organizations transitioning to network-centric environments. In doing so, several related issues such as cost, risks, management and organizational issues must be addressed by the larger software engineering community. In our research, we focus solely on the software architecture related issues of network-centric software systems.

In summary, even though the term network-centric appeared formally in a pure military context, the shift towards network-centric computing was recognized in many non-military domains including academic research. Many argue that the military borrowed the concept of network-centricity from existing business models that software corporations, such as Oracle [16], have developed to integrate its diverse and distributed assets. This is backed by the case studies used in Alberts’ book to discuss the details of NCO vision [12]. Others argue that the concept originated from DOD and has found its way into industry as companies compete for government contracts to develop and provide tools, capabilities, and support mechanisms for NCO. Irrelevant to our discussion is whether the origin is NCO or industry; yet, it is critical that we understand the goals of network centricity within the context of warfighting, and its goals within the context of software-intensive systems.

2.2.4. THE CURRENT STATE-OF-THE-PRACTICE

Since its public inception in 1998, many organizations have embarked on journeys to achieve the ideals of network-centricity. Thus far, we have discussed several perspectives of how network centricity is viewed by various individuals and organizations in both
government and non-government arenas. We have also portrayed a picture that describes the evolution of network centricity as a concept. However, what is crucial is to describe the current-state-of-the-practice.

There are different definitions of what network centricity is. However, these definitions revolve around a common understanding that network centricity is:

- The extent to which a software system has the network as its main axis
- The key to address today's complex computing problems

In the broader context, network-centric computing has the ideal of facilitating the sharing of information and computing resources between locations that have it and those that need it in soft and hard real-time environments. The value of this idea is that it can bring unprecedented success in achieving higher return on investments by order of magnitude. The approach is to make the data, applications, and sometime human resources available on the network.

The following section discusses organizations that have taken this ideal and tried to make it a reality. Sections 2.2 and 2.3 detail two sets of initiatives that have employed the concepts of network-centricity: The first set of organizations includes domain-specific initiatives that have applied the concepts of a network-centric model of operations to their organizational needs. The second set of organizations includes initiatives that have adopted a more collaborative approach to developing network-centric technologies.

2.2. DOMAIN AND ORGANIZATIONAL-SPECIFIC NETWORK-CENTRIC INITIATIVES

To harvest the benefits of network-centric computing, several domain and organizational-specific network-centric initiatives have been instigated in the past few years. The following is a description of some of the more prominent ones.
2.2.1. RAYTHEON’S NETWORK-CENTRIC SYSTEMS

Raytheon is an industry leader in defense and government electronics, space, information technology, technical services, and aviation and special mission aircraft. After the introduction of the network-centric operations concepts and their supporting technologies, Raytheon was among the pioneers to provide such technologies. As a result, a new business branch within the company was established — Network Centric Systems (NSC) [17]. This initiative is dedicated to the development of mission solutions for networking, command and control, and air traffic management among other things. It brings together the core technologies and experience of the company to provide networked systems, net-enabled systems, and net-centric integration to customers worldwide.

Among the many products and services Raytheon has already deployed is the Wide Area Augmentation System (WAAS). WAAS is a satellite-based navigation system (SBAS) for air travel [18]. It is the first operational SBAS that was commissioned by the Federal Aviation Administration (FAA) [19]. Like any network-centric system, WAAS is designed around existing systems and off-the-shelf-components (COTS); this aspect of WAAS is what is called a “system of systems” perspective. However, the system was engineered from the ground up to ensure that it met stringent quality requirements established by the FAA such as interoperability, integrity, accuracy, and availability.

From an architectural perspective, one of the most important features of WAAS is that it has a scalable architecture (Figure 1). The architecture provides coverage area that is matched to the needs of any service provider of air navigation. Another related feature is that WAAS employs a network of unattended reference stations connected to the master station via standard ground communication networks. This feature represents the “network of networks” perspective.
2.2.2. NASA’S INTEGRATING SYSTEMS FOR JOINT OPERATIONS

The National Aeronautics and Space Administration (NASA) has also embarked on the journey towards network-centric operations. NASA’s current quest is to achieve agile, joint operations though the integration of its applications using secure networking [20]. For this purpose, NASA, in partnership with Cisco, is currently involved in defining the next generation network architecture that utilizes Internet protocols (IP) to ensure the interoperability of its terrestrial assets and satellites [21]. The project is a proof-of-concept exercise designed to show the aerospace industry that commercial IP technology is space-worthy. To date, the space community has traditionally used purpose-built hardware for networking. This project represents the first demonstration of a generic commercial network device—a Cisco IP router—onboard a satellite in space [21].

There are several implications that IP-based technologies and hardware can have on NASA’s network-centric joint operations. First, networking capabilities will increase by order of magnitude thereby helping to enable access to unmanned ground stations. Second,
by making satellites as active nodes in space on the Internet, satellites’ ability to interoperate with ground stations and air and space systems will be drastically improved.

Currently, this secure network-centric implementation asset described in [21] uses only a single IP-compliant satellite. However, its network-centric architecture is, by design, scalable to meet the needs of multiple missions, multiple spacecraft, and multiple mission managers. The system mission is distributed across multiple satellites and ground stations, which are accessed via a web-interface for end-users to requests services from those satellites. There have been several successful demonstrations that showed the ability to securely use networks and infrastructure owned and/or controlled by various parties [21].

In this project, NASA focuses on achieving secure network communications, which we believe are an essential aspect of network-centric software systems. NASA focuses on security at hardware level. However, security transcends the network architecture level and lies at the heart of the software itself. To address the security as a quality attribute in a network-centric software system, we have to build security enablers in the software architecture itself as well as its implementation. At the architecture level, security standards and guidelines must be adhered to in order to produce a secure design. In addition, known vulnerabilities should be considered and addressed appropriately in the architecture. In web-based software developments for instance, the OWASP (Open Web Application Security Project) has identified ten top vulnerabilities that an architect needs to be aware of and address in the architecture design.

2.2.3. BOEING’S INTEGRATED BATTLESPACE VISION

The vision of DoD is to leverage information technology and innovative network-centric concepts of operations to develop increasingly capable joint forces. This objective reflects Boeing’s vision of combat in the future, as spelled out by its Integrated Battlespace initiative [22]. Instead of exclusively developing next-generation weapons or stand-alone systems and hardware, Boeing has elected to focus on integrating systems—in effect, designing network-centric applications that bring multiple entities across a wide area into a network where everyone can access the same information.
The Integrated Battlespace is a vision where everyone on the network shares information and communicates to and through each other while following the commander's established rules of engagement. Everyone has the ability to get access to the data needed, to turn that data into information, and to make rapid decisions faster than the adversary can respond [23]. This transformation requires adapting network technology to warfare and reorganizing the classical hierarchical command structure in order to execute these network-centric concepts. Therefore, the transition to network-centric operations not only involves technology but also requires a change in culture, a change in the way things have been done in the past.

Boeing’s approach to network-centricity focuses on achieving a global network technology, which will enable real-time information sharing (a goal of network centricity). At the basic level, they have begun the development of a common network architecture that must be used in all its defense systems. This architecture has common interfaces, common standards, and common protocols. It connects (interoperates) with all nodes, regardless of platform, type of system, or location.

Note: Detailed references, images, architectural documents of existing systems and their architectures can only obtained by Boeing Affiliates and sub-contractors.

2.2.4. SUMMARY

Each of the abovementioned initiatives has a domain focus. However, their approaches to achieve network-centric concepts of operations address several characteristics of network-centric software systems. From a software engineering perspective, the discussion about these initiatives reveals several characteristics that we consider relevant to network-centric software systems. These characteristics are:

- Scalability of both the software and network architectures
- Reliable networked configurations of the network-centric system
- Dynamic behavior that emerges from the behavior of the constituent systems
- Communication through real-time information access rather than point-to-point information sharing
On the other hand, these initiatives also concentrate on other aspects of network-centric computing that are domain-specific or outside the realm of software engineering. These areas are:

- Hardware architectures of systems such as satellites, network devices, etc
- Development frameworks (e.g., .NET versus J2EE)
- Domain-specific tools and technologies (e.g., fiber optics, sensors, nanotechnology)

In the following section, we discuss initiatives that have a broader outlook on the way network-centric systems should be designed.

2.3. INITIATIVES HAVING NETWORK-CENTRIC FOCUS

Besides the domain-specific network-centric initiatives, several other initiatives have been evolved to provide a more general approach to achieving network-centricity. The following is a description of some of the most prominent ones.

2.3.1. ORACLE’S NETWORK-CENTRIC REFERENCE ARCHITECTURE

Oracle Corporation has been a leader in helping corporations build global, network-centric enterprises to compete effectively in the 21st century. Additionally, Oracle is involved in addressing many of the challenges that the military faces in transforming to an information-based warfighting infrastructure. Unlike the initiatives discussed in the previous section, Oracle has a global network-centric vision that stems from the ongoing changing needs of today’s markets [24]. Oracle recognizes that today’s market and product lifecycles are measured in months instead of years, and change is the operative norm. Physical assets are less important than the intelligent use of knowledge. Companies with technology infrastructures and business practices that help them get inside the decision cycle of their competitors have a decisive advantage.

To achieve its vision, Oracle has positioned itself as a network-centric company that has the technology and experience necessary to assist corporations in their transformation to a network-centric model of operations. Oracle used its own model of operations to introduce
a Network Centric Reference Architecture (Figure 2). Even though it is designed to help the military achieve network-centric operations, this architecture provides a comprehensive view that can be the basis for any network-centric software system. In fact, Oracle’s consulting services include the engineering expertise to implement, maintain, and evolve network-centric software systems to meet the emerging requirements for today’s software products.

Oracle’s approach to developing network-centric applications is unique in that it addresses the issue independently from any application domain. Oracle also recognizes the importance of identifying their network-centric solutions at the architectural level because architectures are what drive and guide downstream development. Inherent in their reference architecture are several architectural solutions that are paramount to our discussion of network-centric software systems. These solutions include:

- Multi-level data security that spans all layers of the architecture: Data, Application, and User
- Legacy system integration achieved through a message-based, workflow-enabled integration hub, which provides a wide range of integration capabilities

**FIGURE 2. ORACLE’S NETWORK-CENTRIC REFERENCE ARCHITECTURE [10]**
• Rapid and adaptive development achieved through the layered architecture providing flexibility in terms of evolving the architecture and accommodating the changing requirements [16]

Oracle’s network-centric reference architecture is based on its proprietary technologies such as the 10g Database and Application Server, Collaboration Suite, and e-Business Suite [16]. Additionally, other characteristics are built into the architecture that make it configurable enough to accommodate various implementation technologies, database management systems, and application servers.

2.3.2. NETWORK-CENTRIC OPERATIONS INDUSTRY CONSORTIUM (NCOIC)

The Network Centric Operations Industry Consortium (NCOIC) is a global organization established in September 2004. Its membership is open to all enterprises in quest of applying the vast potential of network-centric technology to the operational challenges facing member enterprises and their customers [25]. NCOIC is committed to integrating existing and emerging open standards into a common global framework to assist with the rapid deployment of network-centric applications [25]. The consortium’s mission is to facilitate global realization of the benefit inherent in NCO. The focus is to enable continuously increasing levels of interoperability across the spectrum of joint, inter-agency, inter-governmental, and multi-national industrial and commercial operations.

Figure 3 represents the technical deliverables of the consortium. Theses deliverables are intended to promote uniform systems engineering methods, tools, and processes among interested developers of network-centric applications. One of the key technical deliverables is the NCOIC Interoperability Framework (NIF). NIF provides the framework for building interoperable network-centric systems and services. It recommends elements of specific domains for widespread use, such as mobile networks, services and information interoperability, and information assurance. Another key technical deliverable that is undergoing development is the Network Centric Analysis Tool (NCAT™). NCAT is a tool that evaluates the ability of a system to operate in a network-centric environment.
The primary value of NCOIC is its ability to assist industry and government technology developers and buyers in changing the way customer systems operate. Catalyzing industry to deliver work products that conform to a common interoperable framework will provide the customers of NCOIC members with an agile operating environment. Done properly, this will translate into a dramatic improvement in the ability of the customer to deliver end-to-end systems unconstrained by service, platform, or system incompatibilities [25].

This initiative is also unique in that it treats the network-centric challenges from a global perspective by bringing together contributions from all parties involved in network-centric computing. The organization has a primary focus on achieving full advantage of NCO but with the goal to extend NCO benefits to non-military domains as well. Among the technical areas NCOIC focus on is the area of Architectures and Standards. This is another major effort that addresses the network-centric issue at the architectural level. We view this as substantiation to our proposal to address the issue of designing network-centric applications at the architecture level.

2.3.3. AIR TRAFFIC MANAGEMENT

For Boeing, network-centric applications also have relevance in the civilian world, most notably in air traffic management. Its vision of revolutionizing the air traffic control is
based on many of the same precepts that guide the network-centric applications sought by the military.

Air traffic is expected to double in the next two decades [26]. Today’s air traffic volume is already straining FAA and aviation resources; existing air traffic management systems cannot continue supporting this increasing demand. As a world leader in aviation, Boeing has taken the lead to resolve this issue through the System Wide Information Management (SWIM) (Figure 4). This solution is based on linking systems together for network-enabled operations. SWIM is an information management architecture for the next generation air transportation systems (NGATS) [27]. It enables shared information across existing, disparate systems for network-enabled operations. Boeing expects that SWIM will dramatically improve the capabilities of existing systems, save capital and operating costs, improve productivity, and offer flexible system expansion.

![SWIM: An Information Management Architecture](image)

**FIGURE 4. SWIM: AN INFORMATION MANAGEMENT ARCHITECTURE [27]**

SWIM is an architecture that is built with network-centricity in mind. It emphasizes several key characteristics of a network-centric system. In particular, SWIM architecture allows for flexible system evolution making it easier to achieve a system of systems solution. In addition, SWIM emphasizes the importance of data and information by providing a common data management infrastructure, which allows for robust information access.
Finally, this architecture has a potentially improved productivity by allowing multiple agencies to collaborate in real-time to handle the increasing demands of air traffic.

2.3.4. SUMMARY

The previous sections in this chapter discuss various initiatives to achieving network centricity. For each initiative, we have listed the focus areas and discussed its perspective on how to develop network-centric solutions. Several of these initiatives have a domain or an organizational approach. Others have a broader approach.

<table>
<thead>
<tr>
<th>Domains →</th>
<th>Military Domain</th>
<th>Industry Domain</th>
<th>Scientific Research</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus areas of all domains ↓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System of systems approach</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Secure networking</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalable physical architectures</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real-time information access</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Standard network architecture</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network-based systems</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Situational Awareness</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interoperability</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Fiber optics technology/Networking</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensors technology</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Legacy system integration</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Multi-level data security</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Rapid and adaptive development</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evolutionary deployment</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Resource management</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Development Frameworks</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Data Schemas Interoperability</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Emergent behavior</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Interdisciplinary collaboration</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

TABLE 2. NETWORK CENTRICITY FOCUS AREAS OF VARIOUS DOMAINS
Table 2 summarizes the areas of focus of each of the domains we looked at in our literature review—military, industry, and the scientific research community. In addition, the table shows the common areas that these domains view as critical to achieving network centricity. There are several common goals that span across the different domains such as the systems of systems approach, while there are others that are emphasized in one domain more than the others are. In order to highlight the common objectives of these initiatives, we only show the area of commonality.

From a software engineering perspective, the discussion about these initiatives reveals several characteristics that we consider relevant to network-centric software systems. These characteristics are:

- System of systems perspective
- Standardization of protocols, processes, and vocabulary for better interoperability
- System decomposition for efficient development and deployment
- Integration of existing and new systems for efficient use of resources
- Real-time data availability and access.

The significance of network-centric computing in solving today’s computing problems have become paramount. We have discussed numerous efforts focused on evolving this field. The question that we ask now is “How far have we come?” The following section briefly gives an idea about the state-of-the-practice of network-centric development.

2.4. STATE-OF-THE-PRACTICE OF NETWORK-CENTRIC COMPUTING: WHERE ARE WE?

Since its formal inception, network centricity has gained a widespread use in government, industry, and academic sectors. It has given rise to initiatives that have focused on the deployment of network-centric solutions to today’s complex computing problems. However, the field is still immature and faces many obstacles its growth toward maturity. The initiatives described in the previous sections take different approaches to constructing network-centric technologies and applications. As a result, their products and deliverables
often exhibit several dissimilar characteristics. The software engineering community is at the point where best practices and processes for developing network-centric applications are being debated. There are several success and failure stories, and there is a significant number of lessons-learned for the abovementioned attempts. However, as we discussed earlier few attempts have been made that focus on offering a global understanding and therefore a common solution to this issue.

Only recently (late 2004) have we seen the inception of the NCOIC [25], which tries to bring a global community to contribute to the formalization of the area. Contributions from NCOIC initiative are still immature and, unfortunately, have not received a widespread use and acceptance from the larger software development community. What we view as a challenge is how this network-centric development model affects the foundational work of software engineering. We have seen through the examples how network-centric computing is fundamentally different from traditional computing. Currently, we have frameworks and models that facilitate the design and implement traditional software. However, what we need are models that help us integrate, augment, and evolve *system of systems*. The basic assumptions of having control over the components of the system are no longer valid. The software engineer now lacks the flexibility to perform a tradeoff analysis on the different architectural alternatives to come up with the best design. Rather, he or she has to work with system members that are outside of his or her control.

The common denominator among the initiatives we discussed in this chapter is that they all recognize the need to communicate, interoperate, and share information and resources. This need place the focus squarely on the network, which is one of the distinctive characteristics of network-centric software systems.

2.5. EVOLUTION OF THE RESEARCH PROBLEM

In the previous sections, we have discussed several initiatives related to developing network-centric systems and applications. The first set of these initiatives have a domain or organizational focus and therefore no comprehensive understanding of network centricity is delivered. The second set of initiatives show a more comprehensive perspective on
network-centric systems but still without a concrete approach on how we can develop them. In both sets of initiatives, there is a lack of coordinated effort to reason about network-centric systems. The result of this is a lack of an integrated picture that helps in understanding the architectural needs to develop network-centric software systems. Consequently, we see a need of a formal approach that helps organize and guide the process of developing such systems.

2.5.1. ISSUES STEMMING FROM THE LACK OF A FORMAL APPROACH

From a software engineering perspective, several issues stem from the lack of a formal approach in characterizing network-centric software systems. First, there is ambiguity in the requirements that specify what a network-centric software system is. Consequently, ambiguous requirements result in a wide range of methods used to design this class of systems. Additionally, having one-sided requirements also affect architectural design artifacts by rendering them non-reusable.

Second, as the majority of network-centric initiatives are domain-specific, we see an absence of a common guide that provides a general solution as to how we should design network-centric software systems.

Third, network-centric applications have only seen a widespread use in the past few years. The majority of the network-centric development efforts have attempted to use traditional technologies and development models to achieve network centricity. Others introduced new technologies. However, using these immature technologies increases the risk on project management. In particular, these technologies are proprietary which narrows the scope of their use and assessment to the correspondent organization.

The issues mentioned in this section are an obstacle to the development of sound network-centric software systems. The list is not comprehensive as we only discussed those issues that are a direct result of a lack of a formal characterization of network-centric software systems.
2.5.2. PROBLEMS RESULTING FROM THESE ISSUES

The issue of ambiguity in the requirements of developing network-centric applications spurs several problems. Most importantly, when we do not know which characteristics are most important to the success of a network-centric application, it is then difficult to achieve those characteristics. Another problem is the complexity of performing validation and assurance if the initial requirements for the system are ambiguous. Finally, the stakeholders hold different views about what the system should do. Communication among the stakeholders becomes a problem if there is no common ground of understanding they can all go back to for reference. Miscommunication among the stakeholders has a negative impact on usefulness and the quality of the system.

As for the issue of guidance, or the lack thereof, when it comes to how we should effectively design network-centric software systems, it becomes extremely important to formalize techniques that allow for such effective design. These techniques can help software developers focus on implementing them rather than being trapped in the loop of try-and-error. Another problem related to the lack of guidance affects the project managers who do not possess reliable and adequate measures to guide the whole development process. Managers have to deal with COTS components, components managed by other entities, interoperability issues, and many others problems.

2.5.3. A KEY BENEFIT OF DEFINING A FORMAL APPROACH

A formal characterization of network-centric systems allows us to be able to identify whether a system is network-centric or not. It also facilitates the recognition of whether an existing individual system has the potential to become part of a network-centric software system. In addition, developers with a common understanding of network-centric software systems will build software that is net-ready, allowing for easier integration and assembly of system of systems. This in turn reduces the cost of fixing interoperability inconsistencies.

Through a comprehensive literature review using our understanding of network-centric systems, we have identified the following system as network-centric: GEON Cyberinfrastructure. The GEON (GEOscience Network) research project is a project that
was created to respond to the pressing need in the geosciences communities to interlink and share multidisciplinary datasets to model the complex dynamics of earth systems [28]. This project is a prime example of how scientists are moving towards increased collaboration to tackle complex problems. As the provider of the software parts of the solution, the software engineering community is on the front lines facing challenges such as predicting earth movements that are responsible for hurricanes, volcanoes’ outburst, and many other natural disasters.

The GEONgrid system is a network-centric computing infrastructure that manages access to remote data collections and services. It consists of a coalition of hardware nodes (system of systems) that are deployed with a standardized GEON software stack. The software infrastructure is based on a service-oriented architecture (SOA) using Web services and Grid Security Infrastructure (GSI) (networked configuration) [28]. The GEON software stack is packaged and tested as a unit to ensure that the components work together reliably, as a single software package. Currently, the GEONgrid spans principal-investigator (PI) institutions as well as a number of other collaborating sites including international sites. Figure 6 shows the underlying architecture for GEON systems.

This architecture (Figure 5) shows how GEON is integrating existing tools, application, and databases that are dispersed worldwide amongst the 22 participating U.S. institutions and many others partners around the globe. The purpose is to create a Cyberinfrastructure that enables geoscientists to engage in questions about earth that were not thought of before [28]. This initiative makes raw rock data from an institution in Indonesia instantly available for a Virginia Tech geoscientist whose specialty includes that particular rock type. This scientist can, in turn, analyze and share the results with the rest of the community for peer reviewing (dynamic runtime behavior) - the interacting resources are not necessarily known at design time but they are dynamically invoked once a task is initiated. The result of such cooperation is a better understanding of natural earth systems. In this type of collaboration, there is a decentralized control over the functionality of the system. All participating institutions have equal access to all resources. The results of bringing together data and analysis tools from around the globe is intended to help in the prediction of natural disasters such as tsunamis, hurricanes, volcanoes’ eruption, and others.
Addressing these kinds of problems has only recently been possible. However, with the use of a network-centric approach, such endeavors are not only feasible, but also achievable.

2.6. SUMMARY AND TRANSITION TO CHAPTER III

Network-centric software systems embody the answer to the pressing need to integrate existing software assets with newly developed applications, and to the growth in size of software-intensive systems that are being developed. They also exemplify a new way of thinking about software systems. Network-centric software systems are the outcome of the inevitable shift from developing a system of statically-distributed resources to a system of dynamically-distributed components that are owned and managed by different entities, and that provide task specific services that can be used to achieve a larger goal. This shift has brought about a need for new architectural approaches to design such systems.

In this chapter, we have identified initiatives that focus on addressing the challenges concerning the development of network-centric software systems. We have also identified the basic principles and characteristics of network-centricity by investigating the roots of the term “network-centric” and its association with the software architecture community.
Further, we have constructed a case for the need for a formal approach that will address these issues.

In the following chapter, we provide a more detailed discussion of the characteristics of network-centric software systems. For each characteristic, we will provide justification as to why we view that characteristic important. In addition, we will enumerate the challenges that face the sound development of network-centric software systems.
CHAPTER III

3. NETWORK-CENTRIC SOFTWARE SYSTEMS: DEFINITION, CHARACTERISTICS, AND CHALLENGES

This research focuses on examining several architectural design issues related to network-centric software systems. Our solution approach consists of two major components. The first component focuses on identifying the characteristics that distinguish network-centric software systems from other large-scale systems. Within this step, we define what a network-centric software system is. We also differentiate between the operational and the structural characteristics of network-centric software systems. Then, we discuss several architectural design challenges related to this class of systems. In this chapter, we focus on this part of our work. The second component of our solution approach, discussed in Chapter IV, addresses the design of network-centric software systems. We do so by identifying the architectural framework within which we can design architectures for network-centric software systems. We then introduce the network-centric architectural style as a solution within this framework.

3.1. DEFINITION OF NETWORK-CENTRIC SOFTWARE SYSTEMS

A network-centric software system is a multi-functional, loosely-coupled, complex system, which more often than not, is an integral part of a larger system that involves the collaboration of software, hardware, and people to solve a complex computing problem. It exhibits several distinctive characteristics. Figure 6 depicts the main characteristics of network-centric software systems.
The first type of characteristics is structural in nature. A network-centric software system has:

- a *system of systems* perspective. Each component system of the network-centric software system is independent and provides a set of functions without being tied to the presence of other component systems.
- an underlying *networked configuration* of its component systems. The component systems communicate among each other over a network. The type of the network (wired, wireless, internet, intranet, etc) varies based on the domain in which the network-centric software will operate, and based on the organizations involved in developing and supporting the individual system components.

The second type of network-centric software systems’ characteristics is operational and includes:

- an *emergent (evolutionary)* behavior. The overall behavior of a network-centric software system emerges from the behavior that emanates from the interactions among the component systems.
• a decentralized control over its data, evolution, and operation. The data is stored in
different data stores, the system’s evolution is manifested through the evolution of
its component systems, and these component systems operate independently from
each other.

The difference between a network-centric software system and any other monolithic large-
scale system resides not only in the structural and operational characteristics but also in
the culture and the perspective behind the development of each. The network-centric
development culture entails the collaboration among various communities to solve
interdisciplinary problems such as national security and business process standardization.
In addition, the perspective behind network-centric computing focuses on the integration
of existing computing capabilities with new ones to address complex computing problems.
In the following section, we use an analogy to make the concept of network-centric
software system more apparent.

3.1.1. DISCUSSION

The analogy between network-centric software systems and monolithic large-scale
software systems is similar to that of buildings and cities [29]. Buildings and
infrastructures, regardless of their size, are monolithic entities that are designed differently
than the way cities are. The design and construction of buildings follow strict rules laid out
in their blueprints by the architect(s). However, the structure of a city emerges over time
through the evolution and the growth of its population and therefore its buildings and
infrastructures. There are certainly guidelines that guide the way a city evolves. Yet, these
guidelines are less strict than what you would see in a building architecture.

In this analogy, we map cities to the concept of network-centric software systems and map
buildings/infrastructures to the concept of a large-scale software systems. The same way a
city is composed of several buildings and infrastructures, a network-centric software
system is composed of several large-scale software systems. The components of a large-
scale, but monolithic, software system (e.g., a cockpit control system) are developed based
on a software architecture designed to achieve a set of functionalities and quality
attributes. The components of such a system are restricted by the architecture and follow relationships that are defined in it. On the other hand, the components of network-centric software systems are loosely-coupled and independently managed. Consequently, the architects must focus on the interface design of the components to ensure interoperability of the components systems. Therefore, we submit that the design of software architectures for network-centric software systems must be based on guidelines governing how the constituent systems interact with one another, similar to a city’s guidelines for construction and expansion. The structure (architecture) of the overall system, then, evolves over time based on the evolution of its components. This finding is the basis for our solution approach, which defines an architectural style that describes the vocabulary and the design elements allowed in the architecture of a network-centric software system.

To show the usefulness of the network-centric concepts, we discuss them within various contexts. In the following section, we describe how network-centric software systems provide solutions to different problems within the military, industry, and scientific computing contexts.

3.1.2. CONTEXTUALIZING NETWORK-CENTRIC SOFTWARE SYSTEMS

As discussed in chapter two, several efforts are underway to develop network-centric software systems. These attempts span the military, industry, and scientific computing communities. In this section, we discuss our understanding and definition of network-centric software systems. We focus this discussion on how network-centric software systems can be used in the DOD operations, industry organizations, and scientific research institutions.

3.1.2.1. MILITARY CONTEXT:

In the military and national security contexts, the availability of the right information at the right time in the right place to the right person(s) can be the difference between life and death. With this understanding, the US Department of Defense has adopted (in principle, at least) a new doctrine of war that focuses on translating information technology into a
warfighting advantage over its adversaries. To achieve this goal, DOD has introduced a plan to transform its operations to become network-centric operations.

Network-centric operations mandate the networking of all DOD branches (Army, Navy, and Air Force) and their associated agencies. The goal is to improve information sharing among these branches to increase the shared situational awareness among all DOD branches. In other words, the DOD plans to increase inter-agency collaboration and information sharing in order to increase its effectiveness in the battlefield, intelligence gathering and interpretation, and national security efforts. In addition, the overhead that occurs due to the command hierarchy is drastically decreased due to the direct access to critical information provided by the network-centric configuration of all DOD systems.

For illustration purposes, we consider the following scenario in an ideal network-centric military environment. A CIA operative in Iraq identifies the names and location of a group of insurgents. Military commanders including soldiers on the ground can immediately access this information through their PDAs. Plans are then made to send a group of marines to raid the insurgents’ building, which is also populated by many civilians. Just before the marines go into the building and start searching, questioning, and possibly engaging the insurgents, new intelligence provides a picture of one of the insurgents and the exact apartment where he is hiding. This information becomes available in real-time to the marines, makes the operation more precise and effective, and helps save innocent lives.

This abovementioned scenario can only be possible in a network-centric environment. Independent component systems (systems of systems) fielded by the different branches of the DOD (decentralized control) collaborate, share information, and interoperate with one another (constitutes emergent behavior) in a real-time fashion to provide soldiers on the ground critical information about the adversaries. This real-time access to critical information (made possible by networking the component systems) is the basis for the shared situational awareness, which the DOD considers the primary enabler of warfighting dominance in the battlefield.
3.1.2.2. INDUSTRY CONTEXT:

In the commercial sector, organizations have realized that the availability of shared information among geographically-dispersed locations directly affects business opportunities. Many organizations have already shifted to a network-centric model of operations. The simplest example we can give to show the usefulness of network-centricity in expanding business opportunities is that of a common everyday transaction. A customer goes to the closest Best Buy store to purchase a widescreen LCD monitor. After the customer chooses the model he wants, the salesperson realizes the model is out of stock in that store and does a search to see if it is available in other Best Buy stores in the area (made possible by networking all Best Buy IT data stores). The salesperson, without telling the customer, informs the customer that the store offers free delivery and that he should receive the package next day.

This scenario, even though it seems simple, would not be as simple if there was not a networked infrastructure in place connecting all Best Buy’s deployed systems (system of systems) in the different stores and warehouses across the country. These systems and databases do not necessarily have to be developed using the same programming languages or have the same data schemas. The important thing is that they interoperate when needed in order to take advantage of every single business opportunity (emergent behavior). In this scenario, the availability of information allows the salesperson to make a business decision without involving higher management (decentralized control). It also makes the transaction process go smoothly and provides the customer with a pleasant customer experience. Therefore, network-centric computing provides a platform for industrial businesses to efficiently utilize their diverse, geographically-dispersed resources to pursue higher business goals.

3.1.2.3. SCIENTIFIC RESEARCH CONTEXT:

In many sciences, collaboration has been a goal and a challenge at the same time. In theory, collaboration between scientists produces good results. With few exceptions, the fruits of collaborative research work are usually much better than individual efforts. However, the means to achieve this collaboration has always been the impediment. With the advance and
the prominence of information and computer technologies, the scientific discoveries we have witnessed in the past ten years equals what we have discovered throughout history. Inter-disciplinary collaboration has become as important as intra-disciplinary collaboration. The full potential of this collaboration can only be achieved through a network-centric model of operations.

The GEON project we have discussed in Chapter II is a prime example of this collaboration [28]. This initiative makes available raw rock data from an institution in Indonesia for a Virginia Tech geoscientist who is specialized in that particular rock type. This scientist can, in turn, make analysis and share results with the rest of the community for peer reviewing and further analysis (emergent behavior). The result of such collaboration is a better understanding of natural earth systems. In this type of collaboration, there is a decentralized control over the functionality of the GEON system. All participating institutions host some but not all the components of the system, and have equal access to all resources. The results of bringing together data and analysis tools from around the globe (system of systems and networked configuration) is intended to help in the prediction of natural disasters such as tsunamis, hurricanes, volcanoes’ eruption, and others. Addressing these kinds of problems has only recently been possible. However, with the use of a network-centric approach, such endeavors are not only feasible, but also achievable.

In this section, we have discussed the usefulness of network-centric software systems. Through different examples, we have shown how network-centric software systems can be useful for military operations, business transactions, and scientific research. The following section discusses in detail the structural and operational characteristics of network-centric software systems.

3.2. KEY CHARACTERISTICS OF NETWORK-CENTRIC SOFTWARE SYSTEMS

In our preliminary research, we have identified at least four characteristics that distinguish network-centric systems from other large-scale systems. We have categorized these four
characteristics into two categories: Structural and operational. In this section, we discuss in detail each of these characteristics.

3.2.1. STRUCTURAL CHARACTERISTICS (STATIC)

Network-centric software systems have characteristics that distinguish them from other large-scale systems. One set of these characteristics is structural in nature. In others words, these characteristics describe how the system’s components are organized. They are concerned with the way the system is structured and the way its constituent systems are decomposed.

3.2.1.1. SYSTEM OF SYSTEMS PERSPECTIVE

Maier defines system of systems as those systems built from components that are large-scale systems themselves [30]. He identified several characteristics related to systems of systems including:

- Operational and managerial independence of the elements: The elements are independently functional systems and are developed and managed independently from the whole system.
- Evolutionary development: The components of a system of systems are not always fully developed but they evolve over time. Therefore, the system of systems itself evolves as its components evolve.

The development of network-centric software systems is guided by the system of systems perspective. The constituent components of a network-centric software system are independent, self-contained, useful systems in their own right. Ideally, these component systems are developed and managed independently from each other. They are brought to work together on a need basis to carry out complex network-centric tasks such as an analysis of rock data that requires analysis tools located in multiple geographically-dispersed sites. Network-centric software systems are systems of systems; however, the opposite is not true. Not all systems of systems are network-centric software systems. The reason is that the former lacks other characteristics unique to network-centric software
systems. For example, an airplane software system is a system of systems in the sense that it is composed of systems that are independently developed to carry out various functions. Yet, these component systems are deployed in a single location (various compartments of the airplane) and come under a central control of the airplane software engineer(s).

In network-centric computing, the independence of the components is often a constraint and not a choice of the designer. Network-centric software systems address complex problems that require the collaboration of independent entities that are often times geographically-dispersed. For instance, the Air Traffic Management system (ATMS) (discussed in Chapter II) is a prime example of a network-centric software system whose components involve independent entities: The software systems used by different airports, weather forecasting agencies, airliners, and emergency first-responders to name a few. These software systems are independent both managerially and operationally. The ATMS evolves over time based on the evolution of these systems. Some of its component systems are replaced, others are upgraded, and some are added.

Based on the ATMS example and the other systems discussed in Chapter II, we conclude that the system of systems perspective is a distinguishing characteristic of network-centric software systems. This characteristic is structural because it is related to how the system components are composed.

3.2.1.2. NETWORK-BASED CONFIGURATION

The second structural characteristic of network-centric software systems is their networked-based configuration. The ability to deploy software over a network has given rise to network-centric software systems. Advancements in network technologies have changed the way we develop, deploy, maintain, and manage effective software solutions. In network-centric computing, a software system is composed of various independent elements that communicate over a network (wireless, wired, Internet, Intranet, WAN, LAN, etc). The reason is that these elements come from various sources and are often geographically-dispersed and owned by various entities.
Network-centric computing is not synonymous to distributed computing. Generally, distributed software systems are developed as centralized systems that run on multiple independent CPUs [31]. Even though these CPUs are connected through a network, they are not network-centric software systems. Performance and scalability of computer processing are usually the drivers behind distributed software solutions, not the integration of and collaboration among independent software applications. Network-centric software systems are capable of operation across a network without an evident centralized control over their elements. However, the fact that network-centric software systems are network-based is not what makes this class of systems unique. In network-centric computing, communications over the network is the solution to the demand of reaching beyond tightly-coupled, local resources to address complex computing problems, e.g., the GEON system. In addition, the distribution of the constituent elements of a network-centric software system is necessitated by the heterogeneity of the sources in which these elements reside. This heterogeneity is manifested in the programming languages in which these components are developed, the data schemas they use, and the platforms on which they run.

Due to the distribution and the heterogeneity of the components of network-centric software systems, in addition to the fact that they are loosely-coupled, the interactions are limited to information exchange. This makes components’ interface design of paramount importance. Organizations developing and managing the constituent components do not have to share proprietary information about their systems; however, they do have to communicate the interface requirements in order to interoperate smoothly with other systems. So, added to the problem of security is the issue of trust (which we will discuss in section 3.3 of this chapter). This means that novel procedures must be put in place to ensure the trustworthiness of any component that can potentially be part of an overall network-centric software system. Doing so at an early stage of design is a daunting task.

In summary, one of the crucial characteristics of a network-centric software system is its networked configuration. This configuration is a structural characteristic, which represents how the constituent components of a network-centric software system communicate.
3.2.2. OPERATIONAL CHARACTERISTICS (DYNAMIC)

Besides its structural characteristics, a network-centric software system also exhibits particular operational characteristics. In other words, those related to the way the system behaves in its operating environment. The first operational characteristic is its emergent behavior that emanates from the interactions of the component systems. The second characteristic is the absence of a centralized control since there is no single component that has control over the overall system. This section discusses these operational characteristics in detail.

3.2.2.1. EMERGENT BEHAVIOR

The behavior of a network-centric software system emerges from the interactions of its constituent systems. The functionalities that the system performs do not (and cannot) reside within any one component system. These functionalities are emergent properties of the network-centric software system [30]. In other words, individual system components are designed to perform (independently) certain tasks. When these system components are configured to work together, the tasks they perform emerge as new functionalities that become properties of the overall network-centric software system, not of any single component system. Therefore, the overall requirements of the network-centric software systems are fulfilled only through the collaboration of the constituent systems. In many cases, these requirements evolve as new component systems are added, upgraded, replaced, retired, etc. This evolution of the system components is what makes the emergent behavior a distinguishing characteristic of network-centric software systems.

The emergent behavior characteristic is a driver for the cultural change needed to begin designing sound network-centric solutions. In this development model, it is difficult to account \textit{a priori} for all possibilities for which a network-centric application can be used. This is due, in part, to the continuous evolution of the components, which drives the change of the underlying functional and non-functional requirements of the system. Therefore, the architectural design process must be at a more abstracted level in order to accommodate the system's evolution and the change in its requirements. In our solution approach, we incorporate this finding when we discuss the types of elements of the network-centric
architectural style. In Chapter IV, we provide a detailed discussion about the network-centric architectural style elements in light of this finding.

3.2.2.2. DECENTRALIZED CONTROL

Our definition of network-centric software solutions emphasizes the fact that this class of systems is complex in many aspects. First, their component systems are independent, fully-functional software systems. Second, their behavior emerges through the evolution of the component systems. Third, their underlying configuration is network-based. The complexity in these aspects mandates that network-centric software systems have a decentralized control. No single component system (or its managing organization) has an overall control over the system. This decentralization occurs at multiple levels. At the data level, information is stored in multiple sources and is provided by different systems. Data has no universal data schema that all data warehouses must follow. The exchange of information has to be done through media like SOAP messages. At the development level, the component system are designed and developed by different teams in different locations. The integration is achieved through a third party whose responsibility is designing the overall architecture of the system. Consequently, management of the system becomes the responsibility of the group through following the general guidelines of evolving and changing the system components.

Decentralized control means that any component is a potential entry point to the system’s functionality. An example of such a system is the Air Traffic Management System (ATMS), which is discussed in Chapter II. In this system, a network-centric task could be initiated by any of the participating systems. A new development in weather forecast can trigger airliners to change their flight schedules. A fire or an accident in an airport could trigger all flights to that airport to be cancelled or routed to another airport, and trigger a call to the first responders units in the area. Such tasks can only be carried out if there is a network-centric infrastructure in place that allows component-initiated interactions among these large software systems. Any attempt to impose any kind of centralized control will affect the very principle of a system of systems perspective, which mandates the independence of the constituent component of a network-centric software system.
After defining network-centric software systems and discussing their structural and operational characteristics in this section, we introduce the architectural design challenges relevant to network-centric software architectures.

3.3. CHALLENGES TO DESIGNING NETWORK-CENTRIC SOFTWARE SYSTEMS

Many challenges that the software engineering community faces are not specific to network centricity. In this section, we focus only on those challenges that are introduced by network-centric software systems. The following are the most dominant ones that we have identified by investigating software systems and software architectures that exhibit network-centric characteristics: Standardization, scalability, on-demand composition, robust-connectivity, security and trust, and test and evaluation.

3.3.1. STANDARDIZATION

The first challenge is the related need to develop software architectures that flexibly accommodate applications and services provided by various developers. An emerging trend in software development efforts is that systems are composed of a mix of local and remote computing capabilities, requiring architectural support that accommodates interoperability, modifiability, and other desirable operational qualities [5]. Thus, we argue that this support should come in the form of an architectural style that facilitates the design of systems using a dynamically-formed coalition of distributed resources. More specifically, new standards (similar to Internet protocols) need to be established for building new components and making the existing ones net-ready.

The widespread use of the internet is due not only to its usefulness but mainly to the standardized format of its protocols. The TCP/IP are protocols that are recognized and used internationally by all internet providers. The internet user needs not know about the details of these protocols to connect to the internet. Similarly, network-centric software engineers need to have standardized design tactics to develop network-centric solutions that will interoperate with other network-centric ready software systems. These standards
should be at a higher level of abstraction so that they are easily understood and adopted in various domains that rely on software systems.

3.3.2. SCALABILITY

The second challenge in developing network-centric software solutions is the need for scalable architectures that can evolve and that accommodate component complexity and variability similar to architecture of the Internet. Network-centric software systems incorporate different components that require different architectural representations and various forms of communication. While many of the existing architectural styles will likely apply, the details of their application will need to change. Thus, we argue that there is a need to define a new architectural style that accommodates these changes.

As an example of an architectural style that will not scale, we use implicit invocation architectural style. This style is a widely-accepted method of designing software systems. Implicit invocation is a style of software architectures in which a system is organized around event handling – broadcasting and subscribing to events. On one hand, this style allows heterogeneous components to be integrated into systems that have low-coupling and high-cohesion, which are two indispensable qualities of any network-centric software system. On the other hand, architects must make assumptions about crucial system qualities such as the reliability of event delivery and routing of messages. In a network-centric model, all such assumptions are uncertain [32]. Therefore, we are further convinced that we need novel techniques that allow architects to design these systems in such a way that it accommodates their dynamic growth.

3.3.3. ON-DEMAND COMPOSITION

The third challenge is the need to develop architectures that enable end-users to form their own system composition. With the rapid growth of internet-based software systems, an increasing number of users want to be in a position where they can assemble and tailor the system they need for their business based on the network-centric system components that are available. Such users may have minimal technical expertise, and yet, will still want a
sufficiently strong guarantee that the parts will work together in the ways they expect [32]. For instance, the Service-Oriented Architecture (SOA) approach enables users to utilize software solutions provided as services, which they can query and employ on a need basis. Similarly, the architecture for network-centric software systems should enable the designers to restructure the component of a network-centric system based on their individual needs. This can only be achieved through modifiable software architectures.

Architects must then find ways to support such needs for network-centric software systems. A network-centric architecture has to encompass characteristics that facilitate the generation of systems that are modifiable and that support an on-demand integration of new components. This challenge must be addressed in conjunction with the standardization issue. The reason is that modifiability becomes an easier challenge to address when we have a standard approach to the design of network-centric architectures that achieve the structural and operational characteristics of this class of systems. This is because the system of systems property will allow users to employ various combinations of some or all of the component systems to carry out their network-centric tasks. In our solution approach, we emphasize the importance of modifiability by describing the style elements at a higher level of abstraction that is not constrained by any programming or development framework.

3.3.4. ROBUST CONNECTIVITY

The fourth challenge that faces designers of network-centric software systems is the need for a robust infrastructure, which supports computing through a large number of independent, heterogeneous, dynamically-integrated components. For instance, the Internet infrastructure supports a broad range of resources such as primary information, communication mechanisms, web applications, services, and many others [32]. A common characteristic among these resources is independence – both operational and managerial. They can join and leave the network at will. They can invoke other resources and can be invoked by others too. Most importantly, they evolve independently from each other. Similarly, a network-centric software system must have an underlying infrastructure that
facilitates a decentralized control over the system elements. Elements are selected and composed based on the task that needs to be carried out.

Due to the intrinsic complexity of automating the selection and composition process, architects must focus on the interface requirements between the elements of a network-centric software system. Within a network-centric model, architects do not necessarily have implementation knowledge about the components that are developed by other entities. They must be guided then by the possibility that any instance of a computation (or a task) is performed by a process defined by dynamically-selected computational entities. In addition, the integration of incorporated components may be infeasible if these components have static interface specifications. For instance, the integration of a component packaged to interact via remote procedure calls with a component designed to interact via shared data can be a difficult task [32]. Therefore, the communication medium between components should be standardized to accommodate dynamic selection based on the task that needs to be carried out.

The challenge of robust connectivity should be addressed in collaboration with the networking community if any standards are to be developed and adopted in a large scale that allows the type of collaboration required for successful network-centric computing.

3.3.5. SECURITY AND TRUST

Security is not a unique challenge to network-centric software systems. However, it becomes significantly more important when combined with the issue of trusting component systems of a network-centric software system without knowledge of their internal implementation details. In traditional security models, the focus is on securing the exchange of and access to information among the interacting components to meet the requirements specification. In the case of network-centric software systems, the managerial and operational independence of the interacting software systems brings about additional security and trust concerns. For instance, if system X is responsible for processing credit cards information and system Y interacts with X to provide credit card information, Y should have enough assurance from X to trust that the credit card
information will not be used for additional processing. Securing the interactions between X and Y is a step towards establishing the trustworthiness of both. In fact, security is the foundation of establishing trust among the interacting components of a network-centric software system. Therefore, security cannot be an added feature to network-centric software architectures; it must be built into the architecture. Software architectures that support the development of network-centric software systems must be designed to have security technologies built into the appropriate elements of the system infrastructure. Finally, additional measures need to be taken in order to establish trust, and that includes communication between the managing entities of the constituent systems of a network-centric application.

3.3.6. TEST AND EVALUATION

The concern over test and evaluation issues is nearly as old as the concept of NCO. In their book on NCO, Alberts, Garstka, and Stein discuss implications of the concept, stating that: “Testing systems will become far more complex since the focus will not be on the performance of individual systems, but on the performance of federations of systems.” This leads to the conclusion that traditional engineering techniques for evaluating network-centric software architectures will not be able to completely meet the network-centric software systems test and evaluation need. Traditional techniques are likely necessary, but by no means sufficient. We believe that they need to be augmented to encompass the complexity of network-centric software systems.

For instance, the SEI’s Architecture Tradeoff Analysis Method (ATAM) is a method for architecture evaluation of software-intensive systems [35]. It emphasizes the participation of the stakeholders of the system in the evaluation process. ATAM has been successfully applied to various governmental and industrial projects, and is now part of the test and evaluation methods in many software development organizations. However, ATAM as it stands now does not necessarily scale to the complexity of network-centric systems exacerbated by the independence of the components. Engaging all the stakeholders from different independent entities that can be located in different parts of the world might be infeasible. Ongoing research by the SEI is underway to investigate the applicability of
ATAM to system architectures and to system of systems architectures. Results from this research should help provide insights on how this challenge can be addressed.

The challenges discussed in this section are unique challenges for network-centric architectural design. Creating an architectural style that facilitates the consideration of these challenges at the architecture level is a reasonable proposition. We do not claim the list of challenges is comprehensive. As the software architecture community expands its understanding of network-centric software systems, we conjecture that more challenges will be identified.

3.4. SUMMARY AND TRANSITION TO CHAPTER IV

The introduction of network-centric software systems poses a challenge to the tradition approach of developing software systems. The main challenge is defining what it means to be network-centric and identifying the distinguishing characteristics of this class of systems. In this chapter, we have provided a definition of network-centric software systems and described the operational and structural characteristics that distinguish them from other large-scale systems. In addition, we have discussed several architectural challenges that surface during the design of network-centric software systems.

In the following chapter, we present the second component of our solution approach. We detail a network-centric architectural style that is intended to be the basis for the design of software systems that are intended to be network-centric.
CHAPTER IV

4. NETWORK-CENTRIC ARCHITECTURAL STYLE GUIDE

The focus of our research is to define a sound architectural solution that guides the design of network-centric software systems. In the previous chapters, we have devised a definition of network-centric software systems, described their operational and structural characteristics, and identified relevant architectural challenges to their design. In this chapter, we examine the perspective from which we address this research problem. Specifically, we describe the level of abstraction of the solution approach. We then employ architectural design principles to introduce a solution in the form of a new architectural style, the *Network-Centric Architectural Style*, which provides guiding principles to the architectural design of network-centric software systems.

To put our solution in perspective, we refer back to the larger problem we are addressing. There is a lack of a common and standardized understanding of what network-centric software systems are and how to design them. Thus, the first step towards reaching that common understanding is to identify a high-level abstraction of what constitute a network-centric software architecture in terms of its primary elements (components and their interactions). Once this step is achieved, then we can begin to identify best practices and technologies that represent these elements. In other words, once there is an understanding of what the key characteristics of network-centric software systems are, what the properties of their elements are, and what challenges need to be addressed, we can then begin to talk about which technologies and practices help us make this understanding a reality. The purpose of the network-centric style is to provide that understanding and serve as a basis for further research to achieve it.
4.1. DEFINITION OF THE NETWORK-CENTRIC SOFTWARE SYSTEMS – REVISITED

A network-centric software system is a multi-functional, loosely-coupled, complex system, which more often than not, is an integral part of a larger system that involves the collaboration of software, hardware, and people to solve a multi-faceted computing problem. It exhibits several distinctive characteristics.

The first type of characteristics is structural in nature. A network-centric software system has:

- a *system of systems* perspective. Each component system of the network-centric software system is computationally independent and provides a set of functions without being tied to the presence of other component systems.
- an underlying *networked configuration* of its component systems. The component systems communicate among each other over a network. The type of the network (wired, wireless, internet, intranet, etc) varies based on the environment in which the network-centric software will operate, and based on the organizations involved in developing and supporting the individual system components.

The second type of network-centric software systems’ characteristics is operational and includes:

- an *emergent (evolutionary)* behavior. The overall behavior of a network-centric software system emerges from the behavior that emanates from the interactions among the component systems.
- *no centralized control* over its data, evolution, and operation. The data is stored in different data stores, the system’s evolution is manifested through the evolution of its component systems, and these component systems operate independently from each other.

The difference between a network-centric software system and any other monolithic large-scale system resides not only in the structural and operational characteristics but also in the perspective and culture behind the development of each. The network-centric
development culture entails the collaboration among various communities to solve interdisciplinary problems as complex as national security or business process standardization. In addition, the perspective of network-centric computing focuses on the integration of existing computing capabilities with new ones to address complex computing problems. In the following section, we discuss our motivation for addressing the problem at the architecture level.

4.2. MOTIVATION FOR A STYLE-BASED APPROACH

Before we begin the discussion about our solution, we present our motivation as to why we address the problem at the architecture level and through the documentation of a new architectural style. In addition, we provide background information concerning relevant architectural principles and terms that we use in our solution.

4.2.1. WHY A SOLUTION AT THE ARCHITECTURAL LEVEL?

Network-centric computing represents a new way of thinking about how to develop useful software systems by integrating software resources using networking technology to respond to the evolving software requirements. The initiatives discussed in Chapter II support this view. The analysis of these initiatives reveals that network-centric computing has two key aspects: A technology aspect and a human aspect (Figure 7). To elaborate, the networks and communication mechanisms used in a network-centric software system correspond to the technology aspect of network-centric computing. Advances in networking technologies are the drive for the spread of a network-centric culture among software developers. This culture is supported by the prominence of new development frameworks such as .NET and J2EE, which facilitate the development of applications that can be deployed over a network. In addition, existing network-centric initiatives show several emerging software engineering technologies used to develop network-centric systems such as Service Oriented Architecture (SOA) and Web Services (WS). On the other hand, network-centric computing exhibits a human aspect as well. Software architects are the ones who make the decisions on how the system’s elements communicate with one another and how they are configured to fulfill the system’s requirements in a networked
environment. Therefore, the system behavior is driven by a human behavior manifested in the architect’s design choices made during the software architecture design. For a network-centric software system, the architect must express an overall structure (architecture) mainly through the specification of the component systems and the communication mechanisms among them. Consequently, we conjecture that the solution approach to designing network-centric software systems must be at the architectural level of the development lifecycle. This approach must provide design alternatives that explicitly address the distinguishing structural and operational characteristics of network-centric software systems.

FIGURE 7. ASPECTS OF DEVELOPING NETWORK-CENTRIC SOFTWARE SYSTEMS

In the following section, we discuss two important architectural principles, viewtypes and styles, in order to set the stage for the discussion about our solution.

4.2.2. VIEWTYPES AND ARCHITECTURAL STYLES

In the previous section, we have established an understanding of the importance of addressing the research problem at the architecture level. In this section, we discuss few
architectural terms, namely *viewtype* and *style*, to create a shared understanding of what we mean when we refer to them in our solution.

First, we start with the term software architecture. The *software architecture* of a system is the representation of its overall structures. This representation is focused on the elements of the systems, the relationships between these elements, and the properties of the elements relevant to the overall goals of the system [36]. To manage the complexities of large-scale systems, these structures are usually captured in several architectural *views* based on various perspectives. An architectural view is a representation of the system from a certain perspective. For instance, a module view shows modules and the relationships between them from a module-based perspective. The Software Engineering Institute (SEI) has coined the term “viewtypes” to represent these perspectives. They have identified three perspectives from which a system can be viewed.

- The *module* viewtype corresponds with the decomposition of the system into constituent subsystems
- The *component and connector* (C&C for short) viewtype corresponds with the System’s behavior and the communication mechanics among its elements at runtime
- The *allocation* viewtype corresponds with how the subsystems are mapped to their underlying environment (hardware, budget, development teams, etc) [5].

A view represents the architect’s way of looking at the system from *one* perspective. That perspective always carries with it a design choice to solve the problem from that perspective. More often than not, these design choices are solutions that the architect has employed in the past and that worked well with similar kind of systems. These solutions are referred to as *architectural styles*. A more concise definition for architectural styles is that they are design solutions to common computing problems. Architectural styles (also referred to as patterns in some of the literature) define elements and relations types that are allowed in an architecture based on that style, along with the constraints on how these elements and relations can be employed [38].
The SEI uses the three viewtypes discussed above to categorize more than 100 architectural styles. The majority of the styles the SEI have categorized falls under the C&C viewtype. The reason is that most of the early research in software architecture focuses on the behavioral aspects of software systems. As a result, many architectural styles have been identified and categorized under the C&C viewtype. For instance, the widely used Client/Server architectural style is a C&C style, where the components are clients and servers, and the connectors are request and reply calls. Other known architectural styles include the SOA style, layered style, decomposition style, communicating processes, and deployment style.

With the understanding of these few terms, we discuss the framework within which we define our architectural style.

4.3. A FRAMEWORK WITHIN WHICH WE DEFINE A SOLUTION

In this section, we present the framework within which we formulate our solution. The design of architectures for network-centric software systems must address their high-level characteristics and the overall structures of how their elements interact with one another. At this level, the documentation of a new architectural style serves as guidance to reason about network-centric software systems. After examining the ways we can document a new architectural style, we use the style guide approach proposed by Clements et al in [5]. The reason is that this approach provides a clear and an understandable format for documenting details and rationales for architectural styles.

As we discussed earlier, the concept of viewtype is used to categorize existing architectural styles. To better understand what a viewtype provides for expressing an architectural style, we present an abbreviated example of the service-oriented architectural (SOA) style guide. Table 3 indicates the type of style information that would be conveyed using the C&C viewtype.
### Service-Oriented Architectural Style Guide

| Elements | Component: Service component  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type: Service</td>
</tr>
<tr>
<td></td>
<td>Component: UDDI Registry</td>
</tr>
<tr>
<td></td>
<td>Type: Registry</td>
</tr>
<tr>
<td></td>
<td>Component: SOAP message</td>
</tr>
<tr>
<td></td>
<td>Type: Remote procedure call</td>
</tr>
<tr>
<td></td>
<td>Component: Invocation</td>
</tr>
<tr>
<td></td>
<td>Type: Invoke procedure</td>
</tr>
<tr>
<td></td>
<td>Connector: Functional component</td>
</tr>
<tr>
<td></td>
<td>Type: Active Process</td>
</tr>
<tr>
<td></td>
<td>Connector: User Interface</td>
</tr>
<tr>
<td></td>
<td>Type: GUI Process</td>
</tr>
</tbody>
</table>

| Relations | Attachment relation associates service interfaces with SOAP message connector.  
|-----------|-------------------------------------------------------------------|
|           | Service interface information is encapsulated as service description in WSDL.  
|           | Attachment relation associates the registry with SOAP messages to access or publish service information. |

| Computational Model | Services are registered in the registry.  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Service invokers acquire registered service information and produce corresponding adapters to invoke services.</td>
</tr>
</tbody>
</table>

| Properties of the elements | Service:  
|---------------------------|---------------------------------------------|
|                           | • Name: should suggest its functionality  
|                           | • Type: defines service type provided as a passive unit that responds to caller |
- **Interface Properties**: depend on the type of the network communication
- **Attachment**: not persistent.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>UDDI Registry:</td>
<td>...</td>
</tr>
<tr>
<td>SOAP Message:</td>
<td>...</td>
</tr>
<tr>
<td>Remote Procedure Call:</td>
<td>...</td>
</tr>
<tr>
<td>Functional Component:</td>
<td>...</td>
</tr>
<tr>
<td>User Interface Component:</td>
<td>...</td>
</tr>
</tbody>
</table>

**Topology**: Each service has at least one adapter. Services can be invoked concurrently or in synchronously depending on adapter organization.

| TABLE 3. AN ABREVIATED STYLE GUIDE OF THE SOA ARCHITECTURAL STYLE |

The SOA architectural style defines a class of architectures that facilitate the design of software solutions by packaging various applications as a set of services that can interoperate to exchange data and provide services to consumers. Web Services can be used to implement the service-oriented architectural style. These web services communicate using SOAP (Simple Object Access protocol) messages. Information about these services can be obtained for a registry called the UDDI, Universal Description, Discovery, and Integration. This information is expressed using the WSDL (Web Service Description language).

Architectural style guides are means to document details about what an architectural style is, what elements it allows, and what interactions and constraints it has. The purpose for the network-centric architectural style is to serve as a reference solution to designing network-centric software systems. We base this reference solution on our understanding of what constitute a network-centric software system, and based on the characteristics that distinguish this class of systems from other systems.

In the following section, we discuss the network-centric architectural style in detail.
4.4. THE NETWORK-CENTRIC STYLE GUIDE

An essential part of documenting a new architectural style is to develop a style guide that records the specialization and constraints the style imposes on its elements and their interactions. This section presents an outline of the network-centric architectural style. We base the documentation of this style on the C&C viewtype structure suggested by Clements et al [5].

The network-centric architectural style provides a common vocabulary that we can use as the basis for reasoning about the design of network-centric software systems. Figure 8 depicts an interaction between two elements of a software system by means of a networked communication, which is an essential characteristic of network-centric software systems.

Node A interacts with Node B over a network. The basis of their interaction is information exchange or sharing. This is because the nodes are independent systems and have little or no knowledge about the implementation details of each other. However, nodes in a network-centric setting can also coordinate their activities to carry out network-centric tasks. Instead of merely exchanging and sharing information, nodes can request or invoke services provided by other nodes. This fact, however, requires that property of minimal coupling among the nodes be preserved.
The core aspect of network-centric software systems is their network-based configuration. So in order to best describe this configuration, we must express these systems focusing on how their elements interact among each other at runtime. Thus, we describe this class of systems from the C&C perspective, which focuses on the components and their interactions at runtime. Consequently, we catalog the network-centric style using the C&C viewtype. Figure 9 shows a summarized style guide for the network-centric architectural style.

The following sections discuss the various components of the network-centric architectural style and their properties.

4.4.1. STYLE ELEMENTS: NODES AND LINKS

Elements are the basic building blocks of a style. Architectural styles have elements that are unique to them. For instance, the client/server architectural style has two element types: Clients and servers. Architectural styles also have guidelines and constraints on how these elements fit together to solve the design problem addressed by each architectural style.
The network-centric architectural style has two main elements: *Nodes* and *links*. We discuss these two elements in detail in the following section.

4.4.1.1. NODES

Like any other C&C architectural style, the elements of the network-centric style are composed of components and connectors. The components are any software system that can be viewed as a computationally independent system regardless of size, functionality, or complexity. For instance, components can be implemented as peer systems, independent client applications, servers, concurrent processing units. We call these components *nodes*.

Nodes in a network-centric software system are independent sub-systems that collaborate to accomplish a common mission, or a business or scientific goal. For instance, an online store requires many collaborating entities to carry out an online transaction. It requires a secure, usable, and reliable web interface that allows customers to shop for products (Node 1). It also needs a credit card processing capability to handle the information of the customer in a secure mode (Node 2). Finally, it needs a shipping and handling component that takes care of placing the items in the hands of the customers (Node 3). This is a simplified version of a business transaction that requires at least three main nodes. These nodes collaborate to carry out the business transaction (network-centric task). It is not necessary that the three nodes are managed by the same organization; however, it is necessary that they can be integrated in various ways to perform their required task that assists in solving the problem at hand. The choice of how they interact is left to the architect. The architect bases their decision on the type of the interacting nodes (servers, peers, services, etc), and on whether the nodes belong to the same institution, or whether they are services obtained from one or more service providers. The nodes work concurrently and interact with one another in various ways, both symmetric and asymmetric. The style specifies no constraints on how these nodes interact from this perspective.

To illustrate how Node 3 (i.e., shipping and handling system), in the previous example, works with the rest of the system, we consider the following real-world scenario (Figure 10). FedEx (a giant shipping corporation) has a new business model in which its shipment
processing system (Node 3) accesses its client’s systems (Node 1) directly for information related to shipping orders. One of FedEx’s clients is HP Corporation. When a new order is created in HP order system (Node 1) and a shipping date is assigned to that item, FedEx shipment processing system accesses that information directly and carries out necessarily transactional tasks such as billing HP, arranging for pickup and delivery, etc. This example describes the essence of network-centricity: Collaboration among and integration of existing independent systems that do not necessarily belong to the same entity. These independent systems, which we refer to as nodes, do not need to share any implementation similarities to collaborate with one another except the fact that they need to exchange information or request services for one another in order for each one of them to perform its assigned responsibilities.

![Diagram of network-centric task]

**FIGURE 10: A SAMPLE CONFIGURATION TO CARRY OUT A NETWORK-CENTRIC TASK**

Interaction among the nodes can be either through information sharing or task invocation. In the previous example, we show how the nodes (1 and 3) share information to carry out their part of the transaction. On the other hand, the interaction between nodes 1 (FedEx System) and 2 (Authorize.Net Credit Card Processing System, for instance) requires a different type of collaboration. The online store system sends information to Authorize.Net
credit card processing system for validation and processing. This interaction requires a task invocation (in this case, a service request to process the credit card) rather than mere information sharing.

Based on the type of interaction between the nodes, different connectors must be employed. There exist at least two types of connectors, which facilitate the interactions between nodes in a network-centric software system. In the following section, we discuss the type of connectors that are allowed in the network-centric architectural style.

4.4.1.2. LINKS: DATA EXCHANGE AND TASK INVOCATION LINKS

Connectors in the network-centric architectural style focus mainly on the network communication type. Different domains use different types of network communications. This often depends on whether software components communicate with hardware components other than those that are hosting them. An example of this kind of hardware is a data sensor and the data store to which it reports the information it senses. Connectors can be requests, replies, invokes-procedure, data exchange protocols, message passing, control, and other types of communication that allow nodes to collaborate within a network in solving a larger task than what each of them can individually handle. We refer to these connectors as links. However, the connector examples we provide do not suggest that we recommend the use of any of them. In fact, we strongly believe that new standardized communication protocols must be available to achieve the full potential of network-centric computing.

The main challenge in the development of network-centric software systems is information exchange and interface compatibility (interoperability) among the different constituent nodes. As we discussed in the previous section, there are different ways nodes communicate with one another. At the architecture abstraction level, we recognize two types of links: Data Exchange Links and Task Invocation Links. Data Exchange links refer to the type of attachment among nodes that requires information sharing and exchange. For instance, to FedEx shipment processing system, HP order system is a mere data repository that it accesses to get information it needs to process the shipments. This is obviously not the only function the HP order system provides. Therefore, the type of link these two nodes
need to interact with each other is a Data Exchange Link. The second type of links is Task Invocation Link. This is best described by the interaction between the online store and the credit card processing system. The online store sends information to the credit card processing system and requests that it processes it. It actually invokes a task that needs to be carried out by another node. Therefore, the type of link they need to communicate is a Task Invocation Link. This kind of links requires assumptions to be made about the nodes being invoked. An example of these assumptions is the parameters that a node needs to send in order for that task to be carried out properly.

In summary, links are the means by which nodes in a network-centric software system communicate with one another. Based on the type of interaction (data exchange or task invocation), a proper type of link is used to facilitate that interaction. The implementation of the communication mechanisms is not mandated by the network-centric style and is left to the discretion of the architect who must consider the problem domain standards and available technologies.

There are suggestions that we can provide here in terms of the kind of communication mechanisms that allow the interactions required by nodes in a network-centric setting. However, we do not suggest or recommend any at this point because we would like to first convey and establish our understanding of what constitute a network-centric software system without being tied to an existing communication technology. We do, however, recognize that employing the network-centric style in the design of an architecture will only help in describing the systems at its highest level of abstraction. Further architectural design efforts must be carried out in order to fine-tune the architecture. The artifact of these efforts could then suggest technologies based on the problem domains and the constraints.

In summary, nodes and links are the building blocks of the network-centric centric styles. The role of the architect is to find the best configuration(s) of the nodes that achieve the overall behavior expected of the system and the quality attributes it must exhibits. More importantly, these configurations must achieve the characteristics of network-centric software systems.
4.4.2. STYLE RELATIONS: DYNAMIC ATTACHMENT

In network-centric architectural style, there is only one kind of relationship between elements: Attachment. In other words, the nodes of the systems must be able to attach to one another for an interaction to occur. The independence and loose coupling properties of the nodes suggest that all nodes are equal and follow no hierarchical layering. The attachment relation is an inherent property of all C&C architectural styles. It suggests that nodes communicate with one another based on the goal they need to achieve and the underlying network used for such communication. The main characteristic of this particular attachment is that it is dynamic. It requires that the nodes interact with one another dynamically to carry out a global network-centric task.

The attachment relation in the network-centric architectural style is realized through data exchange and task invocation links. This attachment specifies which nodes are connected to which nodes via which links. The two types of links constrain the way the nodes interact with each other. This constraint preserves the property of loose coupling of the nodes as well. The more a node needs to know about the implementation of another node, the less they are loosely-coupled. In addition, this constraint requires that nodes have proper interfaces that allow this kind of connection to take place. The nodes interface should exhibit two types of ports: Data Ports and Task Ports. Data ports are interfaces that allow information to be exchanged between two nodes. For instance, a data repository node can have data ports as routines that facilitate SQL queries from other nodes. On the other hand, task ports are interfaces that allow a task to be invoked by another node. A node can have more than one data port and more than one task port. This is obvious since the nodes in a network-centric software system can be configured in various ways to interact with different nodes. Therefore, the type of the interactions varies from one configuration to another requiring different types of ports.

At the abstraction level of architectural design, we do not make any decisions on what kind of protocols are used in implementing the data exchange links, task invocation links, data ports, and task ports. This is left to downstream design work. In addition to reason we have provided, the heterogeneity of the nodes and the tasks they perform require various types
of communication mechanisms that facilitate interaction between loosely-coupled nodes. At this level, we are interested in finding out the right nodes configurations, which allow required network-centric tasks to be carried out dynamically. The network-centric architectural style focuses on addressing the design problem of finding the right structures (the architecture) of network-centric software systems. Implementation details such as communication protocols are left to subsequent architectural design activities.

4.4.3. STYLE COMPUTATIONAL AND COMMUNICATION MODELS

The computational model of the network-centric architectural style focuses on the runtime behavior of the elements. Network-centric software systems are composed of nodes and links. Nodes are connected dynamically based on the current need to carry out a network-centric task. The “system of systems” characteristic of network-centric software systems means the nodes are independent systems themselves, which can be implemented using various architectural styles. We use the term node as abstraction so that it represents components from other existing architectural styles as well as new kinds of component types that might be recognized in the future.

The emergent behavior property is also emphasized in this computational model. Nodes can be integrated and disconnected dynamically, which give the overall system an emergent behavior based on the constituent nodes. Additionally, nodes can evolve over time resulting in an emergent behavior of the overall system.

The communication model specifies how the nodes communicate. Nodes communicate with one another via links. These links can be implemented via different communication mechanisms. Similar to why we use the term node, the term link is abstraction of the communication mechanism and, as such, supports both existing and evolving networking technologies, e.g., IPv6.
4.4.4. PROPERTIES OF THE ELEMENTS

In this section, we describe the properties of nodes and links that need to be documented if the network-centric architectural style is to be used as a design choice for a network-centric software architecture.

4.4.4.1. NODES PROPERTIES

Nodes represent the basic elements of network-centric software architectures. When one documents these architectures, one must consider the following properties.

- **Node Name**: A descriptive name that suggests the function(s) the node provides
- **Node Type**: A characterization of the general function of the node (service provider, processing unit, data store, data provider, etc)
- **Node Description**: Includes the number and type of ports the node has, and the properties required for the node functions to be carried
- **Other properties**: Based on the type of the node, other related properties could include transaction per second, latency, return types, among others.

These are guidelines as to what should be documented in each node in the architecture. The values of these properties vary based on the problem domain and the purpose for which the architecture is used.

4.4.4.2. LINKS PROPERTIES

Links represent the basic elements of communication in network-centric software architectures. The following properties must be considered when documenting a link.

- **Link Name**: A descriptive name that suggests what communication is facilitated by the link
- **Link Type**: Defines the nature of the interaction and the required parameters required for the interaction to take place
- **Other properties**: depending on the network communication type, it may include the protocol of interaction and performance values
The other properties vary based on the nodes involved in the interaction. For instance, interactions involving task invocations require parameters specification, while interactions that involve data exchange require specification about the data schemas and the query-response latency. These decisions are made based on the context within which the interaction takes place.

4.4.5. **Style Topology**

There is no specific topological model to the network-centric architectural style. Nodes connect to and disconnect from other nodes at will based on the network-centric task at hand. However, the closest known topology that can be considered is the mesh topology, which suggests that each node in a network is connected to all the other nodes.

4.4.6. **STYLE SUMMARY**

Table 4 shows a tabular summary of the network-centric architectural style based on the style guide approach.

<table>
<thead>
<tr>
<th><strong>Network-Centric Architectural Style Guide</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Elements</strong></td>
</tr>
<tr>
<td><em>Components type</em>: Independent, active software <em>nodes</em> that have data exchange and task invocation ports (interfaces)</td>
</tr>
<tr>
<td><em>Connectors type</em>: Data exchange and task invocation <em>links</em>, which facilitates communication among the nodes.</td>
</tr>
<tr>
<td><strong>Relations</strong></td>
</tr>
<tr>
<td><em>Attachment</em> relation associates a node’s port with another node port. Port compatibility must be enforced.</td>
</tr>
<tr>
<td><strong>Computational Model</strong></td>
</tr>
<tr>
<td>Nodes are configured dynamically to collaboratively carry out network-centric tasks. A task can be initiated by any node in the system.</td>
</tr>
<tr>
<td><strong>Communication Model</strong></td>
</tr>
<tr>
<td>Nodes are connected by means of links between them. A node can communicate with multiple nodes via proper port-link combinations.</td>
</tr>
<tr>
<td>Elements Properties</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Link:</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

| Topology            | There is no topological model to this style. Nodes connect and disconnect to other nodes at will. The system has a “changing” topology that is unrestricted and that is based on the task being carried out. |

**TABLE 4. THE NETWORK-CENTRIC ARCHITECTURAL STYLE GUIDE**

The network-centric architectural style consists mainly of nodes and links that facilitate communications among the nodes (Figure 5). The analogy of its application is that of a dynamic data structure. When we define a dynamic data structure such as a linked list, we do not know what an instance of that data structure will look like at runtime. However, when we design this data structure, we can define how new nodes can be added and removed. Similarly, the network-centric architectural style allows architects to identify the overall structure of how nodes work together, and what kind of links can be used for nodes to interact with one another at runtime. The network-centric style also helps in moving critical design decisions to the level of architecture, which in turns makes it possible to address certain quality attributes and to perform risk analyses to avoid misallocation of resources.
After the discussion about the network-centric style, an important question remains to answer. The question is “how does the style address the characteristics of network-centric software systems.” In the following section, we provide an answer to this question.

4.5. HOW DOES THE STYLE ADDRESS NETWORK-CENTRIC CHARACTERISTICS?

In the previous section, we have introduced the network-centric architectural style as a beginning for addressing the issue of design with network-centricity in mind. However, we have only made few references to how the style actually satisfies the characteristics of network-centric software systems. In this section, we describe the network-centric architectural style with respect to the network-centric characteristics.

4.5.1. SYSTEM OF SYSTEMS PERSPECTIVE

Network-centric software systems are systems of systems and, as such, they comprise sets of independent component systems that are systems in their own right. The network-centric architectural style refers to these component systems as nodes. We use the term node to connote two main points:
• The first point is to emphasize the independence of the component systems in terms of their computational operation.

• The second point is that a component system is a member of a set of component systems. In other words, the term *node* connotes that there is a network of nodes. This then emphasizes that component systems are part of a network of systems.

Although the size and complexity of the nodes in a network-centric software system vary, they are all considered first-class citizens. At design time, the abstraction of elements in terms of various equally-righted nodes allows the architects to focus on the main important design problem—that is to find out the how these nodes interact with one another to carry network-centric tasks that are paramount to the complex computing problem the network-centric software addresses.

### 4.5.2. NETWORK-BASED CONFIGURATION

Network-centric software systems comprise nodes that collaborate over communication network(s) to achieve common goals. These network-based interactions among the nodes are paramount to what this class of systems is set to do. The network-centric architectural style comes under the C&C viewtype and, as such, focuses on the interactions among the nodes of the system. These interactions are achieved through links. Links represent the connectors that facilitate interactions among the nodes over networks. They specify the type of interaction two nodes in the system use to exchange information or invoke tasks.

### 4.5.3. EMERGENT BEHAVIOR

Network-centric software systems exhibit emergent behavior. This means that the overall system behavior emanates from the interactions of the nodes. This behavior has two dimensions. The first dimension is reflected through the different configurations of the nodes, which results in behavior that cannot be shown by any single node. The second dimension of the emergent behavior property results from the change in behavior of the nodes due to adding new components, and deleting or modifying existing ones.
In a network-centric software architecture, the elements are loosely-coupled nodes that communicate with one another through links. Every node in the architecture can communicate with any other node. Therefore, when a set of nodes need to carry out a network-centric task, they can actually initiate the appropriate set of interaction to perform that task. Decomposing elements into nodes also allows for the emergence of new behavior over time. The independence and the loose coupling of the nodes make it easy to add new nodes and modify/delete existing ones.

4.5.4. NO CENTRALIZED CONTROL

One of the important characteristics of network-centric systems is that they do not have centralized control. In other words, no single component has control over the system. The network-centric architectural style does not go against this property. This choice is dependent on the decisions the architect makes. Ideally, the architect must focus on interface design and the type of links that realize those interfaces. In addition, the decomposition of the functionality should respect the loose coupling property. This is particular important if the system is to be built from the ground up. In reality, network-centric software systems are composed of existing systems. The architecture of the new system is needed, but re-architecting the nodes go against the premise of this class of systems—that is the collaboration of existing, independent systems to accomplish a bigger goal than what each can accomplish on its own. In this style, the assumption is that it be used to organize the component systems. It provides no guidelines as to how to architect the individual systems.

4.6. SUMMARY AND TRANSITION TO CHAPTER V

Chapter IV discusses the details of our solution approach. To address the issue of network-centric design, we have first identified the framework within which we can address the issue. We have analyzed ways software engineers have gone by to develop network-centric solutions. The conclusion is that a good design solution must be at the architectural level and must be at a higher abstraction level than what existing solutions provide. Since network centricity is concerned with certain type of systems with particular properties, we
have decided to use the style guide approach suggested by Clements et al [5] to document a new architectural style to define this new class of systems.

The network-centric architectural style is a solution approach to the problem of identifying sound architectural design of network-centric software systems. The style employs two main elements, namely nodes and links, and provides guidelines about how these elements can be applied to achieve a network-centric architecture. Additionally, the style dictates a higher level of abstraction that is required to reason about network-centric software systems making it possible to achieve the characteristics of network-centric software systems. Finally, we have shown how the network-centric architectural style addresses the key characteristics of network-centric software systems.

Chapter V summarizes our work and discusses our plans to continue exploring the areas of architectural design for network-centric software systems. Particularly, we are interested in finding out how the Capabilities Engineering approach can contribute to better design of network-centric software systems. We are also focused on expending our understanding of how to develop this class of systems from the Requirements Engineering perspective. Namely, we ask the following question:

- Can we identify the architectural relevant requirements before we start the design phase?”

If this is possible, then we want to investigate how.
CHAPTER V

5. SUMMARY, CONTRIBUTIONS, AND FUTURE WORK

In this chapter, we present a summary of the different sections of this thesis. We also enumerate the contributions of our research work to the software engineering body of knowledge. Finally, we discuss the vision that guides our continued research to address further software engineering challenges related to the development of network-centric software systems.

5.1. RESEARCH SUMMARY

This thesis work addresses the issue of architectural design of network-centric software systems. The emerging concepts of network-centricity have been accompanied by several initiatives that focus on how to develop network-centric software systems. Our preliminary analysis of these initiatives shows that they are either domain-specific or less comprehensive in addressing the issue of how one can architect these systems. Therefore, we have proposed a formal characterization to what constitute a network-centric software system. Then, we have developed a new architectural style that guides the architectural design of this new class of systems. Figure 12 shows the steps of our solutions approach.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Classify and define network-centric software systems based on network centricity concepts</td>
</tr>
<tr>
<td>2</td>
<td>Identify the structural and operational characteristics of these systems</td>
</tr>
<tr>
<td>3</td>
<td>Identify relevant challenges pertinent to their architectural design</td>
</tr>
<tr>
<td>4</td>
<td>Investigate ways to achieve the characteristics and address the challenges</td>
</tr>
<tr>
<td>5</td>
<td>Select the software architecture phase as the level at which this problem should be addressed</td>
</tr>
<tr>
<td>6</td>
<td>Document a new architectural style based on the style guide approach</td>
</tr>
</tbody>
</table>

FIGURE 12. STEPS OF OUR SOLUTION APPROACH
In Chapter I, we describe some of the symptoms of the research problem emphasizing the fact that there are diverse interpretations of how to develop network-centric software systems. These often-conflicting interpretations have resulted in different approaches to achieving network centricity. In addition, we define the research problem and identify the issues pertinent to addressing it. These issues include the lack of a unified understanding of what constitute a network-centric software system. Finally, we describe the steps of our solution approach, which leads to the development of the network-centric architectural style. First, we propose a succinct characterization of what constitute a network-centric software system including the key operational and structural characteristics of this class of systems. Then, we use the style guide approach to develop and document the network-centric architectural style.

Chapter II details the background material and motivation behind this research. It presents our analysis of the impact of the emerging paradigm shift in developing large-scale software systems. This shift is characterized by the move from platform-centric development to network-centric integration and evolution of software solutions. Chapter II also contains a literature review about the progression of the network-centric concepts, initiatives, and technologies. This review includes initiatives from industry, military, and academia.

Within Chapter III, we propose and substantiate our definition of what constitute a network-centric software system. We discuss, in details, the key operational and structural characteristics of network-network software systems. The second component of this chapter deals with the issues pertinent to the architectural design of this class of system. We identify and discuss several of these issues including standardization, scalability, and interoperability among others.

Finally, Chapter IV introduces several architectural concepts such as software architecture definition, architectural styles, and viewtypes. Chapter IV also includes the specifics of our solution approach manifested in the network-centric architectural style guide. Additionally, we describe the components of this new architectural style, as well as how this style addresses the key operational and structural characteristics of network-centric software systems.
In summary, we can divide our thesis work into two major components. The first component deals with defining network-centric software systems and identifying their distinguishing characteristics. Within this step, we differentiate between the operational characteristics and the structural ones. In the second component of our solution approach, we focus on the architectural design issues for this class of systems. We do so by identifying the architectural framework within which we can design architectures for network-centric software systems. Then, we introduce the network-centric architectural style as a solution within this framework. To document our solution, we use the style guide approach to provide a concise high-level documentation of the network-centric architectural style.

5.2. CONTRIBUTIONS TO THE SOFTWARE ENGINEERING COMMUNITY

We present two major contributions from our research in the area of architectural design of network-centric software systems.

A. A formal characterization of network-centric software systems

Our preliminary research work reveals a lack of a common understanding of what a network-centric software system is. To address this issue, we have surveyed the current network-centric systems and initiatives and identified the common denominator characteristics among all these initiatives. Employing the basic principles and goals of network centricity, we have devised a characterization for network-centric software systems that includes distinguishing operational and structural properties for this class of systems. As a contribution, our characterization enable software engineers focus on various software engineering issues related to the development of network-centric software systems. It provides a common ground for reasoning about how to better design, implement, test, and deploy network-centric solutions.

B. Network-centric architectural style:

Our second major contribution in this research effort comes from addressing the issue of how to develop network-centric software systems. In particular, we investigate how we can design network-centric architectures that enable the realization of the key operational and structural characteristics of this class of systems. Instead of using existing architectural
approaches, which are not necessarily intended for network-centric software development, we develop a new architectural style to guide the architectural design for network-centric software systems. The network-centric architectural is unique because it allows architects to describe architectures for network-centric software systems at a higher level of abstraction that enables the incorporations of existing and emerging technologies. The elements of this new style preserve the properties of independence, emergent behavior, and loose coupling, which are major characteristics of network-centric software systems. However, even though the network-centric architectural style is applied at a higher level of abstraction, it does provide direction as to how to address various quality attributes such as maintainability and modifiability.

Our continued research focuses on other areas within the development lifecycle for network-centric software systems namely the area of requirements engineering. In the following section, we describe the different venues where we can take our continued research.

5.3. FUTURE WORK

In this research work, we address the issue of architectural design for network-centric software systems. We propose a solution approach that employs the network-centric architectural style to create high-level network-centric software architectures. We have two potential areas of research in which we can have our continued research work. These areas are capabilities-driven architectural design and integration of existing technologies to provide more guidance on how to apply the network-centric architectural style. Figure 13 shows these two research areas.
CAPABILITIES-DRIVEN ARCHITECTURAL DESIGN

Our solution assumes that the requirements engineering process produces sound functional requirements as well as key quality attributes for the system being developed. These functional and non-functional requirements are the input to the architecting phase. Since network-centric software systems are unique in the way they are developed (integration instead building from scratch), and the purpose for which they are developed (collaboration among communities of same interest), the requirements generation process should be adapted to reflect the needs of this new class of systems. Consequently, we envision many research opportunities and questions yet to be answered in this area.

Through our involvement with the capabilities engineering research, we have identified a link between capabilities engineering and architectural design activities, particularly in identifying the high-level components of the system from a functional standpoint. We believe that sound functional decomposition resulting in capabilities can help better the architectural design activities (Figure 14) by providing highly-cohesive and loosely-coupled elements that can be readily put together to achieve quality attributes such as change tolerance, modifiability, and maintainability. However, the issue that remains is whether capabilities can serve as elements of the architecture. In other words, is there a direct mapping from capabilities to elements of the architecture? If the mapping is not one to one, there is a need to either further decompose certain capabilities or group them together. In this case, another issue arises: What is the impact of the decomposition and grouping (if required) of capabilities on the loose coupling and high cohesion properties of
these capabilities? As a consequent, we believe that capabilities engineering can be tailored to incorporate architectural considerations related to other quality attributes such as performance and security. This tailored process can produce better slices (capabilities) that can be readily used in the architectural design.

FIGURE 14. CAPABILITIES AS AN INPUT TO ARCHITECTURAL DESIGN

INTEGRATION OF EXISTING TECHNOLOGIES TO SPECIALIZE NETWORK-CENTRIC ARCHITECTURAL STYLE

The network-centric architectural style must be used to describe the system at a higher level of abstraction. This is particularly important to achieve the four key characteristics of network-centric software systems. Another reason is to enable the incorporation of various technologies, in particular communication technologies. However, once the initial architectural activities achieve these key characteristics, further architecting is required to provide more guidance to downstream design and implementation. Continued research in this area should focus on surveying the existing technologies and best practices and choose the ones that are better suited for the development of network-centric software systems. For instance, currently the eXtensible Markup Language (XML) and the SOAP are technologies that have enables applications-to-applications communication. They are widely used and becoming the community standards for network-based communications. Further research in this area can explore the advantages, limitations, and the constraints
such technologies put on the development of sound network-centric software systems. Based on this research, recommendations of best practices and suited technologies can be critical contributions to the state-of-the-art.
REFERENCES


36. ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems


[86]
VITA

CONTACT INFORMATION

Amine Chigani
achigani@vt.edu
www.aminechigani.com
3160 Torgersen Hall, Virginia Tech, Blacksburg, VA 24061

EDUCATION

Hassan II University, Casablanca, Morocco (2000)
AA, English Literature

Radford University, Radford, VA (2003)
BS, Computer Science (major), Mathematics (minor)

Virginia Tech, Blacksburg, VA (2007)
MS, Computer Science and Applications

EMPLOYMENT

Virginia Tech, College of Engineering, Graduate Teaching Fellow (Fall 2007 – Present)

Virginia Tech, Geosciences Department, Research Assistant (Summer 2004-Fall 2005)

Virginia Tech, Teaching Assistant (Summer 2005, Spring 2006)

webMethods, Inc., Security Intern (Summer 2006)

Virginia Tech, Graduate Assistant (Fall 2006, Spring 2007)

Dish Network, Inc., Technical Support Representative (Spring 2004)