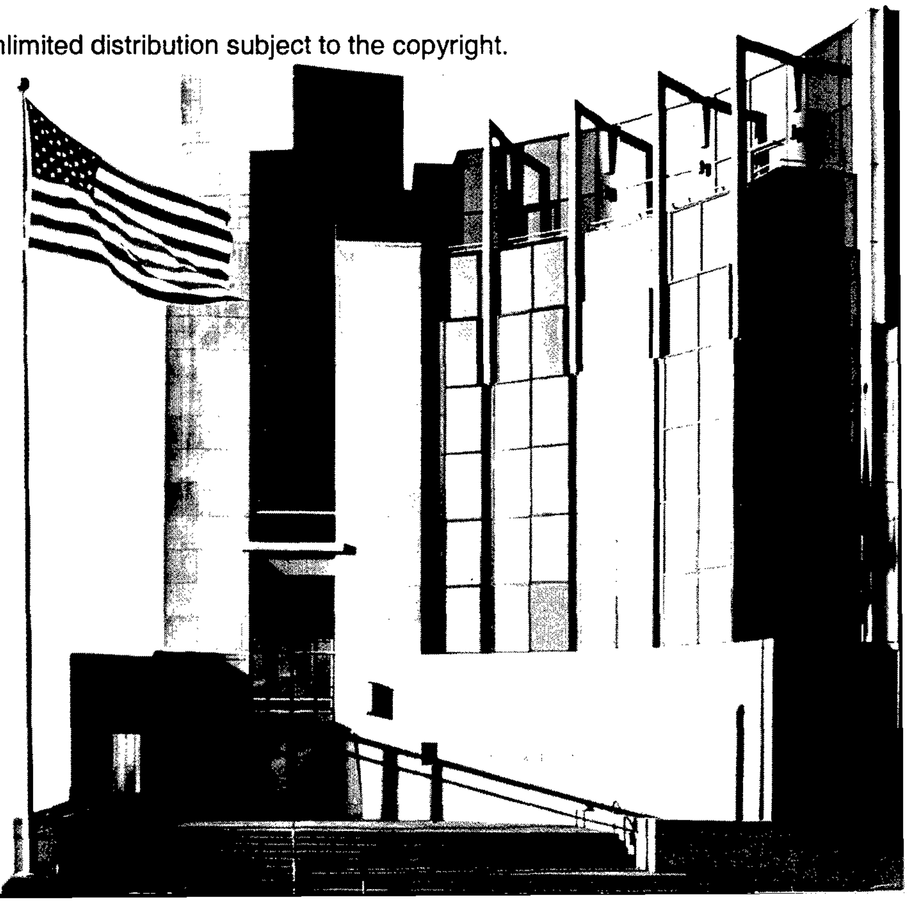
Carnegie Mellon
Software Engineering Institute

SMART: The Service-Oriented Migration and Reuse Technique

Grace Lewis
Ed Morris
Liam O'Brien
Dennis Smith
Lutz Wrage

September 2005

Integration of Software-Intensive Systems Initiative



Technical Note
CMU/SEI-2005-TN-029

SMART: The Service-Oriented Migration and Reuse Technique

Grace Lewis
Ed Morris
Liam O'Brien
Dennis Smith
Lutz Wrage

September 2005

Integration of Software-Intensive Systems Initiative

Technical Note
CMU/SEI-2005-TN-029

Contents

List of Figures

Abstract

This report describes the Service-Oriented Migration and Reuse Technique (SMART). SMART is a technique that helps organizations analyze legacy systems to determine whether their functionality, or subsets of it, can be reasonably exposed as services in a Service-Oriented Architecture (SOA). Converting legacy components to services allows systems to remain largely unchanged while exposing functionality to a large number of clients through well-defined service interfaces. The U.S. Department of Defense (DoD) is adopting this approach by defining SOAs that include a set of infrastructure common services on which organizations can build additional domain services or applications. SMART considers the specific interactions that will be required by the target SOA and any changes that must be made to the legacy components. An early version of SMART was applied with good success to assist a DoD organization in evaluating the potential for converting components of an existing system into services that would run in a new and tightly constrained DoD SOA environment.

1 Introduction

Today's business environment, whether commercial, government, or military, is characterized by rapid change. From the commercial standpoint, traditional bricks-and-mortar retailers are adapting to the increased competition from online sellers. Manufacturers are responding to increased competition from low-cost providers. Other firms are divesting to refocus on core capabilities, or are acquiring to fill gaps in capabilities. From the standpoint of government, the rapid pace of change is shaking up even the most tradition-rich organizations. For example, according to U.S. Postmaster General John E. Potter [Potter 05],

> *...Structural changes in societal and business communications have altered the economics of our business model over the past four years ... First-Class Mail volume has declined 5.4 percent while advertising mail volume has grown 6.1 percent (Statement before U.S. House of Representatives Subcommittee, April 26, 2005).*

Competition from email, fax, and online bill paying is eating into core first-class mail business.

From the DoD perspective, change is constantly required to keep ahead of adversaries whose own capabilities are enhanced by the "trickle down" of new technologies. The DoD is likewise forced to adapt to the exigencies of many diverse enemies, operating environments, and tactics, rather than the single, monolithic (but predictable) adversary of the cold war.

As the business environment changes, so must the accompanying computer systems that organizations rely on to fulfill their missions. Creating computer systems that can be assembled efficiently, adapted quickly to changing conditions, and easily maintained across broad enterprises presents perhaps the greatest set of challenges for the software engineering community. To do so without abandoning previous investments in software systems seems almost too good to be true.

With the advent of universal network availability and distributed systems, standards and technologies that provide better abstractions for code, and new models of interoperability, many experts believe these challenges can now be met through *Service-Oriented Architectures* (SOAs).

1.1 Service-Oriented Architecture Definitions

At the core of an SOA is a *service*. A service is a coarse-grained, discoverable, and self-contained software entity that interacts with applications and other services through a loosely coupled, often asynchronous, message-based communication model [Brown 02].

Several of the terms used in this definition require further explanation:

- *Coarse-grained* refers to the tendency of services to provide significant business process capability, as opposed to low-level business functions. While nothing prevents an organization from implementing low-level functions as services, concerns such as efficiency and performance normally make such an approach impractical.
- *Discoverable* refers to the fact that services can somehow be located and their interfaces understood. This is often misinterpreted as requiring dynamic (runtime) discovery, but such is not the case and mechanisms for discovery may exist to support the programmer in identifying services prior to runtime.
- *Self-contained* refers to capabilities that do not require context or state information of other services, nor do they maintain state from one request to another.
- *Loose coupling* refers to a design principle whereby modules have few, well-known dependencies, and interfaces to the module are defined to be as independent as possible from the implementation of the module. This allows modules to be independently deployed, and encourages the construction of applications that make no assumptions about service implementation beyond the characteristics present in the well-defined, published service interfaces. Ideally, the service implementation can change without affecting service users so long as the service interface is unchanged.

An SOA is a collection of services with well-defined interfaces and a shared communications model. A system or application is designed and implemented to make use of these services. This developed capability may itself provide services within the overall SOA. Figure 1 provides a high-level view of an SOA that presents several options for incorporating services:

1. Service interfaces are added to existing enterprise information systems for applications to use, while these systems remain unchanged for internal users.
2. Service-specific code is written to provide functionality for applications to use.
3. Services written by third parties and deployed elsewhere are used within applications.

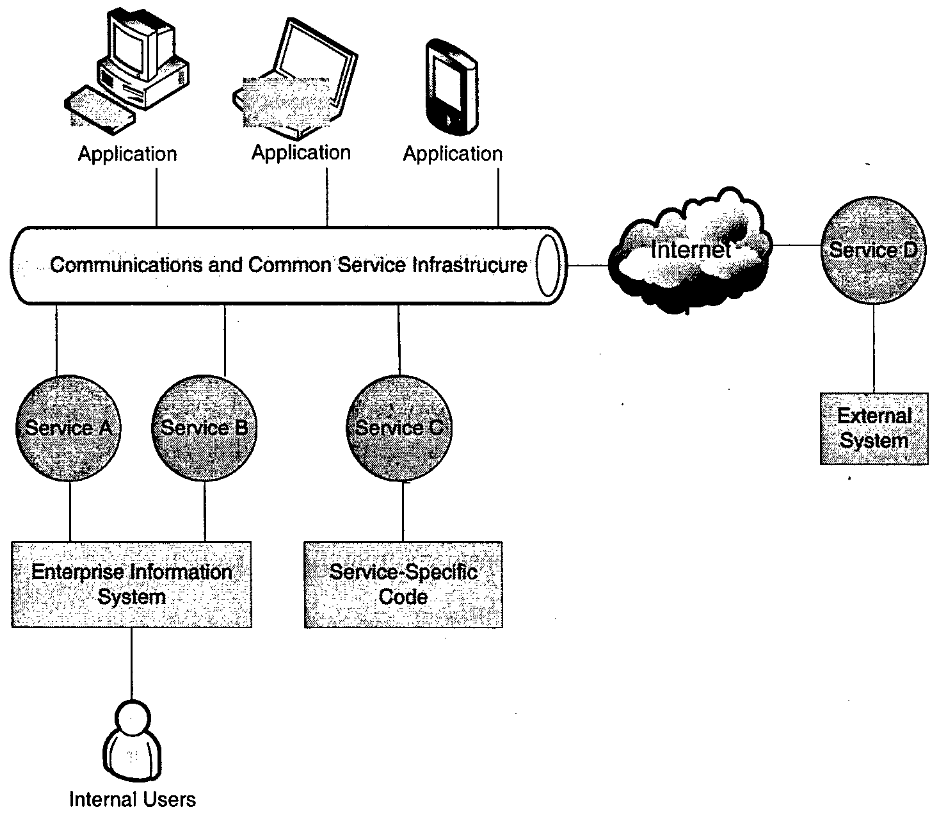


Figure 1: High-Level View of Service-Oriented Architecture

The most common form of SOA is that of Web services in which all of the following apply: (1) service interfaces are described using Web Services Description Language (WSDL), (2) payload is transmitted using Simple Object Access Protocol (SOAP) over Hypertext Transfer Protocol (HTTP), (3) Universal Description, Discovery and Integration (UDDI) is optionally used as the directory service [Lewis 05]. However, WSDL, SOAP, and HTTP are not the only foundation on which an SOA can be built. Other technologies such as CORBA and IBM's Websphere can be used as part of the messaging backbone of an SOA. The DoD is incorporating proprietary technologies to develop SOAs that function in its highly secure and demanding environments.

1.2 Migration to SOA

Momentum is growing within business, government, and defense communities to migrate software systems toward SOA. It is easy to see why the SOA topic is "hot," since proponents suggest a long list of advantages:

- simple standards that define the available interfaces and structure of data that is conveyed across those interfaces
- platform and language-independent interfaces based on these standards, which allow applications to invoke services operating on any device supporting the SOA regardless of the hardware platform, operating system, or implementation language

- clear separation of service interface from implementation, thus allowing many service upgrades to occur without impact on service users
- message-oriented communication allowing distribution across a wide area
- loose coupling between services, minimizing interdependencies and facilitating reuse
- mechanisms for discovery of services available and for establishing connections with services

Clearly, the hallmark of SOA is flexibility. Computing platforms and languages can vary; services can be accessed across a network via simple, well-defined interfaces, and (presumably) without concern for side effects resulting from dependencies between services. This allows applications to use (or be composed of) services efficiently and effectively.

However, migration to SOA is neither easy nor automatic, particularly when the SOA will execute within a tightly constrained environment. Consideration must be given to all three parts of the SOA identified in Figure 1:

- The Communications and Common Service Infrastructure Provider must identify the network and communications protocols and standards to be employed. He or she must also determine what additional SOA infrastructure capabilities are necessary and provide them as common services (e.g., service registry, service orchestration mechanisms).
- The Application Designer must develop application-specific code and locate/select appropriate services to be used by the application, or develop code to invoke mechanisms to select services. He or she is concerned with whether services invoked by the application meet a full range of capability, quality of service, and efficiency of use expectations.
- The Service Provider must identify a needed service, and modify or develop service code to provide a useful capability to the widest range of applications possible.

1.3 Legacy Systems and SOA

SOAs also offer the promise of enabling existing legacy systems to expose their functionality as services, without making significant changes to the legacy systems themselves. This is one of the most attractive features of SOA to many organizations that do not wish—and cannot afford—to walk away from their investment in legacy systems or redevelop the same capabilities as services from scratch.

Enabling a legacy system to interact within an SOA, such as a Web services architecture, is sometimes relatively straightforward—this is a primary attraction to the approach for many businesses. Web service interfaces are set up to receive SOAP messages, parse their content, invoke legacy code, and optionally wrap the results as a SOAP message to be returned to the sender. Many modern development environments provide tools to help in this process, and commercial organizations are rapidly employing these environments to expose their business processes to the world.

However, characteristics of legacy systems, such as age, language, and architecture, as well as of the target SOA, can complicate the task. This is particularly the case during migration to highly demanding and proprietary SOAs such as those being proposed for many DoD systems. In these cases, it may not be immediately obvious how best to use legacy code—or even whether to use it. DoD (and similar) migrations to SOAs will likely rely less on semi-automated migration, and more on careful analysis of the feasibility and magnitude of the effort involved.

Migration to SOA, although promising, will be a daunting challenge for organizations like the DoD. The size of such organizations, the number of distinct groups involved, the diversity of the software, and the overall scope of the effort conspire to destroy consistency and introduce complexity to the migration effort. What the DoD and similar organizations need is a systematic process that addresses a wide range of considerations in order to achieve consistent results in making decisions regarding which legacy components should migrate to provide services within an SOA, and how that migration should occur.

This report focuses on assisting the large group of maintainers of legacy applications that are trying to determine whether their components can be retrofitted to provide services within an SOA. Section 2 outlines a process developed at the Software Engineering Institute (SEI) for evaluating legacy components for their potential to become services in an SOA. Section 3 discusses the pilot application of this process on an actual project. Section 4 provides conclusions and discusses next steps.

2 The Service-Oriented Migration and Reuse Technique (SMART)

The Service-Oriented Migration and Reuse Technique (SMART) was developed to assist organizations in analyzing legacy capabilities for use as services in an SOA. SMART was derived from the Options Analysis for Reengineering (OAR) method developed at the SEI that was successfully used to support analysis of reuse potential for legacy components [Bergey 02].

SMART gathers a wide range of information about legacy components, the target SOA, and potential services to produce a service migration strategy as its primary product. However, SMART also produces other outputs that are useful to an organization whether or not it decides on migration. SMART input (from documentation and interviews) and output are depicted in Figure 2.

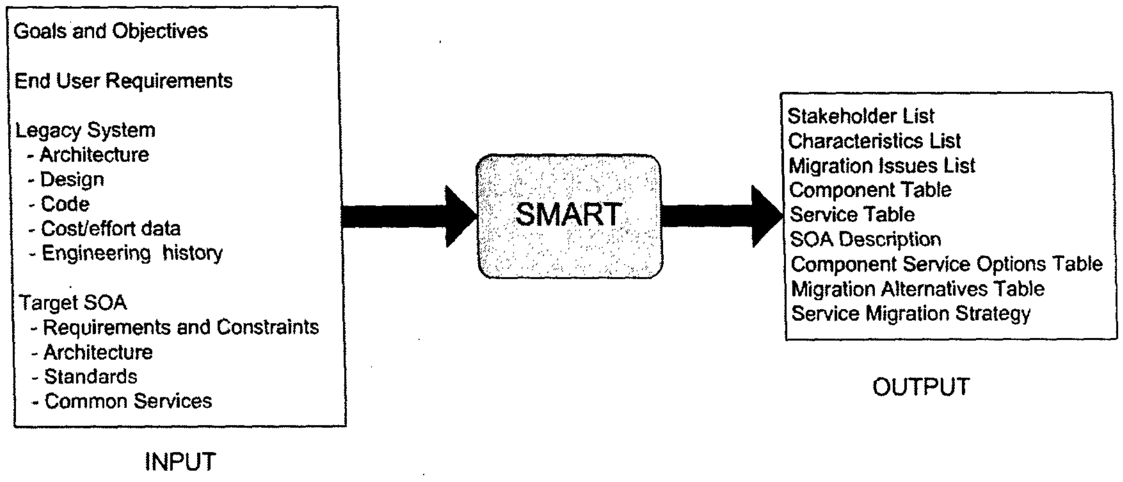


Figure 2: SMART Input and Output

SMART consists of five major activities, each divided into several tasks. The activities and generalized process and information flows of SMART are depicted in Figure 3. However, the number of artifacts considered, the time required, and the specific activities of a given application of SMART depend on previous activities and expectations of the requesting organization. For example, if the requesting organization has specific legacy components in mind for migration, SMART activities will be focused on those components.

Information for the first three activities (on the left) is gathered during an initial orientation meeting and through several additional meetings between the SMART team and the organization tasked with the migration activities. During these meetings, the SMART team assesses stakeholder needs, identifies the SOA vision, and elicits a high-level description of the architecture and other features of the legacy system (as listed in Figure 2). Available documentation is gathered for the legacy system in general, for legacy components that may be transitioned to services (if previously identified), and for the target SOA. In some cases, the target SOA may not be complete, so SOA documentation may describe a future state.

Information-gathering activities for the first three activities are directed by the Service Migration Interview Guide (SMIG). The SMIG contains questions that directly address the gap between the existing and target architecture, design, and code, as well as questions concerning issues that must be addressed in service migration efforts. Use of the SMIG assures broad and consistent coverage of the factors that influence the cost, effort, and risk involved in migration to services.

It is not necessary for the team to complete all data gathering during these initial activities. Additional opportunities are provided during the Analysis activity.

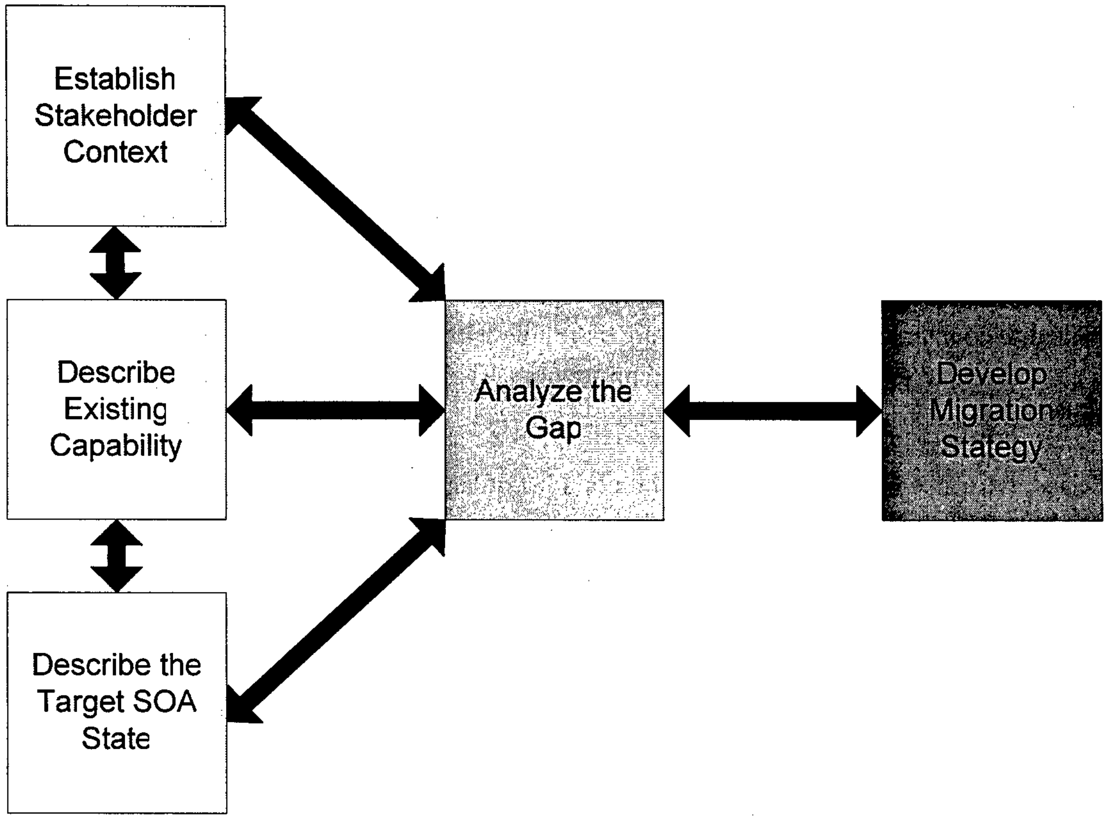

Establish Stakeholder Context
Describe Existing Capability
Describe the Target SOA State
Analyze the Gap
Develop Migration Stategy

Figure 3: SMART Activities

SMART is not strictly sequential. The order in which the activities are performed can vary. For example, it may be convenient to elicit information about the target SOA prior to learning about the existing components. In addition, information gained during any SMART activity may lead to reconsideration of previous activities. For example, a particular migration strategy may require that new stakeholders be consulted. Or, information gathered during the initial activities may be insufficient to determine whether conversion to services is cost effective. In this case, additional information about the component and/or target SOA may be gathered during the analysis process. This proved to be the case in the pilot application of the SMART process described in Section 3, where a hands-on analysis of the legacy code was used to answer specific questions and verify assertions.

The five activities and associated tasks of SMART are detailed in Sections 2.1 through 2.5. A summary of the output from SMART is provided in the appendix.

2.1 Establish Stakeholder Context

In order to establish the context in which the migration to services will take place, the SMART team employs the SMIG to solicit information about stakeholders. Stakeholders typically include the owners and current end users of the legacy system, and the potential end users of the migrated services operating within the SOA. Other stakeholders who are sometimes important include those who are funding or controlling the migration effort, groups defining the target SOA, and Verification and Validation groups that will certify the properties of the new services.

The key to this activity is to identify who knows most about the legacy system, what it currently does, and what it should do as a service or set of services. A significant but non-obvious goal is to identify the parties who are best situated to indicate whether there is sufficient demand for the service to warrant migration efforts. Input from these parties is critical to counteract any tendency toward assuming without evidence that the legacy system is a good source for useful and appropriate services. Their input will also influence the interface design for the resulting services.

This activity also initiates the construction of a list of legacy component characteristics that will later drive the analysis process. A list of migration issues is also begun.

The Establish Stakeholder Context activity has three tasks:

1. Create Stakeholder List.

   The Stakeholder List identifies stakeholders and the type of information to be elicited from each, and provides contact information.

2. Create Characteristics List.

   The Characteristics List identifies information about components that will be gathered and later considered to determine whether service migration is feasible and appropriate. The Characteristics List may be updated at any time during the process as additional relevant features are identified.

3. Create Migration Issues List.

   The Migration Issues List identifies concerns that will have to be addressed during the migration process. Some migration issues may be applicable to all components and services (i.e., general issues), while others may be applicable only to specific components or services.

The SMIG contains questions that will guide the capture of information related to:

- Goal of Migration
- Expectations
- Potential Service Users
- Legacy System End Users and Owners
- Contractors
- Legacy Components and Potential Services

2.2 Describe Existing Capability

The goal of the second activity of SMART is to obtain descriptive data about the components of the legacy system. The activity employs the SMIG to gather data about a specific set of topics related to the legacy system, but the SMART team has the latitude to pursue interesting leads. For example, the SMART team may ask questions about the philosophy and strategies applied for use of COTS products in the legacy system on learning that the system developers opted to use a custom (non-standard) interface to a commercial database.

Basic data solicited during this activity includes the name, function, size, language, operating platform, and age of the legacy components. Technical personnel are questioned about the architecture, design paradigms, code complexity, level of documentation, module coupling, interfaces for systems and users, and dependencies on other components and commercial products.

In addition, data about the relative quality and maturity of legacy components is gathered, including outstanding problems, change history, user satisfaction, and likelihood of meeting longer term needs. Historical cost data for development and maintenance tasks is collected to support effort and cost estimates.

The Describe Existing Capability activity has three tasks.

1. Update the Characteristics List.
2. Create a Component Table.
   - Identify components under consideration for migration to service.
   - Capture characteristics (identified in the Characteristics List) of each component under consideration as a column in the Component Table. These columns indicate the information that will be gathered on each component.

3. Update Migration Issues List.

   Identify and capture any general migration issues, as well as issues specific to components identified in the Component Table.

The SMIG contains questions that will guide the capture of information related to

- legacy system characteristics
- legacy system architecture
- code characteristics

2.3 Describe the SOA State

The third activity of SMART is intended to

- gather evidence about potential services that can be created from the legacy components
- gather sufficient detail about the target SOA to support decisions about what services may be appropriate and how they will interact with each other and the SOA

Initial information about potential services often comes via SMIG-directed conversations with legacy component owners. However, the information gathered must be tempered by data from users, corporate architects, domain groups, communities of interest, and reference models that address service definition. In some cases, these groups and models will define the entire set of services that support the organization's goals, and into which any potential services built from the legacy components must fit.

The characteristics of the target SOA will temper decisions about whether legacy components can be reused. The degree to which a legacy component is inconsistent with these characteristics will profoundly influence the overall migration costs.

Note that the target SOA can be owned by the same organization that owns the legacy components, or by another organization. It may provide a fixed or pre-existing architecture, or the architecture for the SOA may be developed simultaneously with the reengineering of legacy components. The actual placement along this spectrum will have important technical and political consequences for decisions that are made.

The Describe SOA State activity has four tasks:

1. Update the Characteristics List and Component Table.
2. Create SOA Description.
   - standards and technologies to be employed and relevant guidance documents
   - execution platforms, substrates, and middleware
   - deployment requirements
   - special requirements regarding handling of data and state

- specific quality of service requirements that affect potential services directly and end-to-end[1] quality of service requirements that affect related collections of services

3. Create Service Table.
    - Identify potential services that can be derived from components as well as any services that may already have been identified by the organization. This table will be expanded during subsequent activities.
    - Capture information regarding potential services in each Service Table entry.
        - Information sources include groups such as potential service users, corporate architects, domain groups, and communities of interest.

4. Update Migration Issues List.

    Update the Migration Issues List with general, component- or service-specific information as necessary.

The SMIG contains questions that will guide the capture of information related to

- service requirements
- target SOA and legacy system adaptation
- service-oriented changes
- support

2.4 Analyze the Gap

The goal of the fourth activity is to identify the gap between the existing state and the future state and determine the level of effort and cost needed to convert the legacy components into services. This analysis may also suggest potential tradeoffs between the target architecture and the legacy components. For example if the target SOA is flexible, or if it is still in the process of being defined, a relatively minor change to its requirements may allow more legacy components to be converted to services or may simplify the conversion effort. However, substantial risks to the migration effort are introduced when the target SOA has a large number of to-be-defined areas.

SMART uses several sources of information to support the analysis activity. The issues, problems, and data gathered as the SMART team investigates the available components, required services, and SOA requirements form one source of information. A second, optional source of information involves the use of code analysis and architecture reconstruction tools to analyze existing source code. Where documentation is insufficient or where there is uncertainty about code characteristics such as dependencies on commercial products, tool analysis is very helpful. This option can also be used with great effect to survey representative portions of the code to verify other opinions and judgments.

[1] By *end-to-end*, we mean the pathway through applications using cooperating services and the network to perform a specific task.

The tasks associated with the Analyze the Gap activity include

1. Create Component Service Options Table.
    - Compare and map available components (Component Table) to needed services (Service Table). Mapping need not be one to one. A component may need to be split into several services, or several components may be combined to form a service. Also, it is possible to have several ways of producing a service. This may involve alternate ways of using available components, as well as using capabilities that were not part of the original system, such as COTS products.
    - Capture each mapping option in a Component Service Options Table entry.
2. Identify Additional Data Needed.

    If there are gaps in understanding, the SMART team identifies the information and creates a strategy to obtain it. For example, if there is concern about the completeness and accuracy of the data gathered, the SMART team may elect to gather additional data via hands-on architectural and code analysis.
3. Gather Additional Data.
    - Execute strategy to gather additional data.
    - Update Component Table, Services Table, and Component Service Options Table as appropriate.
4. Analyze component/service options.

    Using the information contained in the Component Table, Services Table, Component Service Options Table, and Migration Issues List,

    - Estimate the cost and effort required to migrate the component(s) to services, and to build the services from scratch, for each entry in the Component Service Options Table.
    - Determine the level of difficulty and risk associated with the migration effort.
    - Update Component Service Options Table.

2.5 Develop Migration Strategy

The final activity of SMART involves recommending one or more of the options documented in the Component Service Options Table, selecting a strategy to achieve the goal, and presenting the SMART team findings. In many cases, the migration strategy may involve multiple steps, such as an initial "quick and dirty" wrapping, followed by restructuring of the application (now service) into appropriate layers, and finally by modification to use other services. Example elements of a strategy include

- the identities of specific components to migrate
- recommendations regarding the ordering of migration efforts
- specific migration paths to follow (simple wrapping vs. rewriting of code)

- identification of increments that lead to increasing capability
- suggestions regarding organization(s) best equipped to lead the migration effort
- suggested coordination with related efforts (for example, SOA infrastructure builds)

The tasks associated with the Develop Strategy for Service Migration include

1. Select recommended component/service options.

   Update Component Service Options Table with recommendations

2. Create the Migration Alternatives Table.

   For each recommendation in the Component Service Options Table, there may be more than one viable strategy to achieve the migration goal. These strategies may vary along many dimensions, such as the components selected for migration, the sequencing of migration activities, the use of external services, and the types of modifications made to the code. Viable strategies are documented as entries in the Migration Alternatives Table.

3. Analyze entries in the Migration Alternatives Table and select a strategy.

   Considering risk, cost, effort, schedule and other relevant factors, select a Service Migration Strategy.

4. Prepare and Present Findings.

   Prepare a final presentation detailing the Service Migration Strategy.

SMART provides a preliminary analysis of the viability of migrating legacy components to services, migration strategies available, and the costs and risks involved. In particular, it attempts to answer several questions:

- Which components can reasonably be used to derive services?
- What sorts of activities must be performed to accomplish the migration?
- What strategies are most appropriate for the migration effort?

The sponsoring organization receives a detailed briefing of the results of SMART, but the briefing is not intended to replace system engineering activity. It is assumed that the organization will reflect on the results and pursue further engineering analysis along the lines recommended by SMART.

3 Pilot Application of SMART

An early version of SMART was applied in a recent analysis of the potential for migrating a set of legacy components from a DoD command and control (C2) system to a target SOA. This early version differed from the current structure of SMART because the SMIG and various outputs had not been formalized. However, similar concepts were applied informally.

3.1 Establish Stakeholder Context

Stakeholder context was established through a meeting with the government owners of the system and the contractors who had developed the system. At the initial meeting, the SMART team was given an overview of the set of systems, the history of the systems, the migration plans, and the drivers for the migration. The team was also given a brief orientation to the target SOA and provided with system documentation.

DoD systems have recently focused on the concept of network-centric operations: to provide forces with access to integrated information from a variety of previously unconnected sources [Alberts 00]. This focus requires strong emphasis on interoperability to ensure that systems work together effectively. To facilitate such interoperability, the DoD has initiated a number of projects that examine different aspects of the infrastructure for network centric operations. Several of these projects are developing SOAs so that C2 applications can be built as a set of interactions between infrastructure services (e.g., communication, discovery) and services that are specific to the C2 domain (application domain services). Current and future DoD program offices have been targeted to contribute application domain services.

The owners of the systems recognized that a selected set of components from their C2 system, if converted to application domain services (ADS), would have broad applicability. They had targeted potential services as part of their initial analysis of ADS requirements. The SMART team's role was to perform a preliminary evaluation of the feasibility of converting a set of their components into these application domain services.

3.2 Describe Existing Capability

To determine the existing capabilities of the C2 system, the SMART team met with the contractor and representatives of the government to focus on a limited number of legacy components and to select characteristics for further screening. These sources provided significant detail about the legacy system, but the available architecture documentation was incomplete. In particular, logical and development views of the system architecture were not available. This represented a problem for our analysis, and is discussed in more detail in Section 3.4.

As detailed by the contractors and government representatives, the pilot C2 system has two parts: (1) a mission planning system and (2) a mission execution system that adds situational awareness to the planning capability. These two systems were initially developed as part of a product line. Both rely on a set of core components for the data model, data analysis, and visualization.

A physical view of the current system is illustrated in Figure 4.

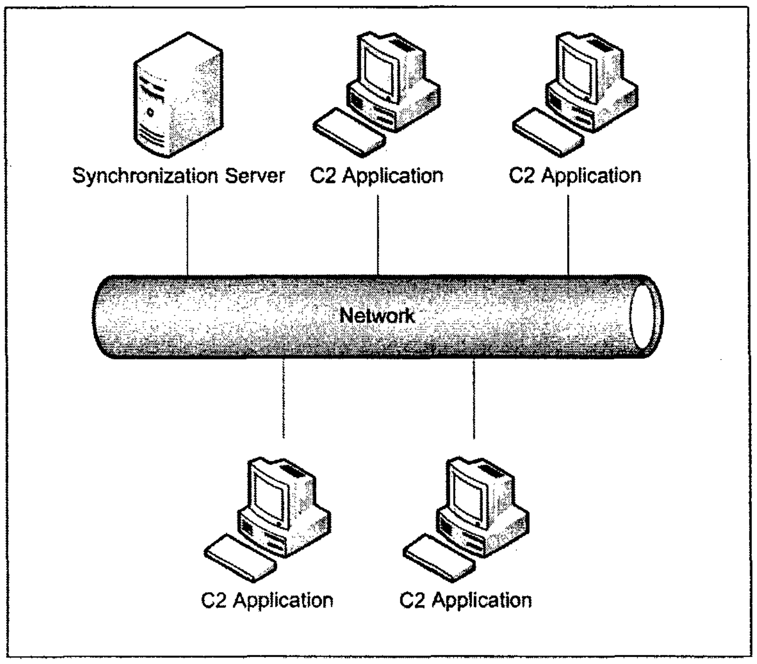


Figure 4: Physical View of the Current System

As shown in Figure 4, in the current environment there is a single instance of the C2 application per machine and an instance of the synchronization server deployed on another machine. Instances of the C2 application interact with the synchronization server to send and receive data updates. The entire system is under the control of one organization. The SMART team used this observation at several points during the study to help the client understand the implications of making the transition from the current environment to an SOA environment.

The current system, written in C++ on a Windows operating system, had a total of about 800,000 lines of code and 2500 C++ classes. In addition, the system had dependencies on a commercial database and a second product for visualizing, creating, and managing maps. Both commercial products have only Windows versions.

The team focused on the 29 specific C++ classes that would presumably provide the basis for the seven potential services that the government team had previously identified, and that offered high probability of providing useful insight. The team identified characteristics that would be the focus for analyzing the components, starting with those provided by OAR and supplemented with team knowledge of the necessary characteristics of services operating within the target SOA. The characteristics included the following:

- size
- complexity
- level of documentation
- coupling
- cohesion
- number of base classes
- programming standards compliance
- black box vs. white box suitability (i.e., wrapping vs. making internal modifications)
- scale of changes required
- commercial mapping software dependency
- Microsoft dependency
- support software required

These characteristics formed the basis for the more detailed analysis discussed in Section 3.4.1.

3.3 Describe the SOA State

The system owner had completed a preliminary identification of potential services that could be built from components of the legacy system. This analysis was derived from high-level requirements for applications that were being targeted as consumers of services to be provided by the SOA. The system owner had matched legacy functionality to these high-level requirements and provided some initial estimates of the contents of the potential services.

The SMART team investigated the target SOA through an analysis of available documentation and through a meeting with the developers. The target DoD SOA is currently under development. It is being built using a variety of commercial products and standards, along with a significant amount of custom code. The effort is focused on satisfying a number of specific quality attributes important to the DoD, such as performance, security, and availability. In order to meet these needs, the SOA will impose a number of constraints on potential services. Because the SOA is still under development, the specifications for how to deploy and write services are still unclear.

The target SOA is illustrated in Figure 5.

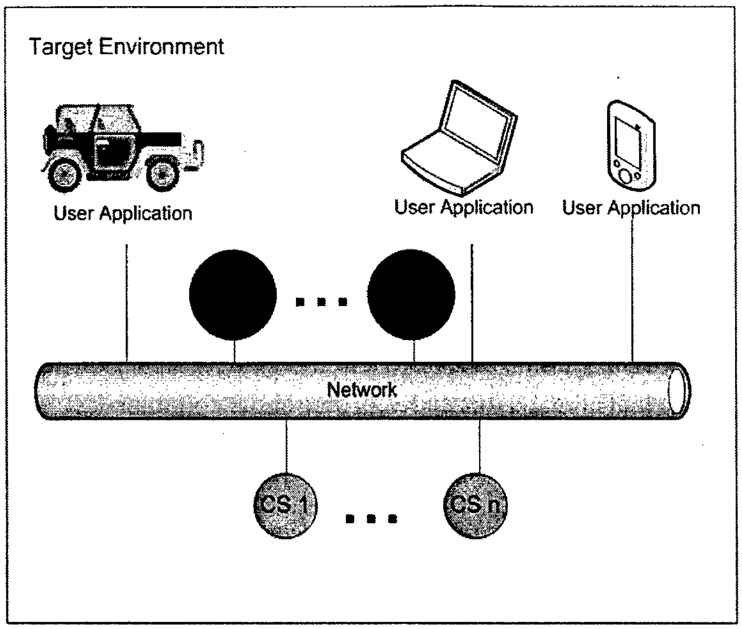


Figure 5: High-Level Physical View of the Target SOA

Figure 5 shows that the SOA includes common services (CS) that are to be used by user applications and application domain services (ADS). The SOA owns the interfaces for the common services. The environment allows for a set of ADSs that will derive their requirements from user applications. It is still unknown if ADS and CS services will run within a single machine or will be distributed over a network. Groups within the DoD are invited to submit proposals for services to meet these requirements, either by building them from scratch or by migrating them from legacy components. These requirements then need to be analyzed in detail and matched to existing functionality to determine what can be used as-is, what has to be modified, and what requires new development.

Even though the full details of compliant services for the SOA have not yet been worked out, the SOA imposes a number of constraints on organizations that are developing ADSs from legacy components. Some of the constraints/requirements for developers of ADSs include

1. An ADS must be self-contained, that is, it should be able to be deployed as a single unit.

   In this specific target SOA, services must be stand-alone and of small granularity so that they can be deployed as needed on standardized and often limited-resource platforms. In a legacy component, functionality that has been identified as part of a service needs to be fully extracted from the system, including code that corresponds to shared libraries or the core of a product line.

2. In the target SOA, an ADS must be deployable on a Linux operating system.

   For Windows-based legacy components this could be a problem, especially if there are dependencies on the operating system through direct system calls or if there is a dependency on commercial products that are only available for Windows systems. Ideally, system calls should be eliminated. If this is not possible, they should be evaluated to see if there are equivalents in the Linux operating system or if this functionality is part of one of the common services.

3. All services will share a common data model and all data will be accessed through a Data Store common service.

   The need for a common data model is driven by a desire for information to be shared and understood by all user applications. As a result, services will no longer define internal data. All data will be defined as part of the common data model. Legacy components must replace all dependencies on databases and file systems with calls to the data store service and ensure that all the data needed is part of the common data model.

4. An ADS will use the Discovery common service to find and connect to other services.

   If the ADS will rely on other services, code to discover and connect to these services will have to be written. Once the service is developed it must be advertised. This is done by registering the service with the discovery service. Once this advertised service has been registered, other applications that wish to use this service will perform a discovery on the available services and choose which service(s) they desire to use.

5. An ADS will use the Communications common service for communicating with other services.

   The target SOA provides tools for generating data readers and data writers that will take incoming and outgoing data and format it accordingly.

3.4 Analyze the Gap

Given the known and projected constraints of the target SOA, the SMART team analyzed the legacy components to determine their suitability for reuse as services, and the amount of effort and risk that would be involved.

The SMART team performed three different types of analyses: (1) an analysis of the changes to the legacy components that would be necessary for migration to the SOA, (2) an informal evaluation of code quality, and (3) an architecture reconstruction to obtain a better understanding of undocumented dependencies. The results of these analyses allowed the team to define a service migration strategy that mitigated some of the risk caused by the instability of the target SOA. These analyses are each described in the following subsections.

3.4.1 Changes to Legacy Components

The team analyzed the candidate legacy components in terms of the characteristics that were developed in Section 3.2. The SMART team identified dependencies of the selected classes on other classes, the commercial mapping software, the commercial database, and Windows, but was not sure that all dependencies had been identified. Most of the legacy documentation was in the form of code comments and from a tool *DOxygen* which can extract after-the-fact data from the C++ code, such as classes, attributes, dependencies, and comments. However, during the analysis the team found that the DOxygen tool only picked up first-level dependencies. This indicated that the coupling and the amount of code that was used by each class was higher than could be estimated from the existing documentation.

There were also no consistent programming standards, leading to idiosyncrasies in the code produced among different programmers. This increased the difficulty of our analysis, and it would also increase the difficulty of any reuse. As might be expected from a relatively recent, object-oriented system, overall cohesion was found to be high. The contractor provided estimates for converting the components into services, based on a set of simplifying assumptions on the actual make-up of the target SOA and the final set of user requirements.

A summary of the initial analyses of converting the selected components to services is shown in Figure 6. Base classes are those from which the classes in the service are inheriting properties in the object-oriented context. "Coupled" classes are those that contain code that is used by the classes in the service. It is important to account for these, as they represent code that must be migrated.

Services		Number of Services	Number of Classes	Size (LOC)	Migration Effort (MM)	Level of Difficulty	Level of Risk
Scope of Analysis	Selected Services	7	16	16,163	12.5	Low to Medium	Low
	Base Classes		4	2,199	4	Low to Medium	Low
	"Coupled" Classes		3	5,388	1	Low	Low
	Total		23	23,750	17.5	Low	Low

Figure 6: Results of Initial SMART Analysis

Using the existing contractor, the level of difficulty of making these changes would be low to medium, and the risk would be low because of the contractor's familiarity with the systems. However, because of inadequacies in the architecture documentation and the contractor's underestimation of the amount of code used by the potential services, there remained a number of gaps in understanding of the system. For example, it was mentioned that one of the services made extensive use of the data model. This data model had over 1000 classes and was used by almost every class included in the potential services. Even though analysis did not initially focus on the data model, because of its size it now represented the largest potential source of reuse in our study. However, as pointed out in Section 3.3, constraints of the target SOA may not allow the data model to be reused.

As a result, it was not possible to accurately know how many other classes are used by a specific service. In addition the estimates for rehabilitation of the legacy components would have been understated. For example, the calls to user interface code would have to be removed, and it would be necessary to know where these are located.

To get a better understanding of these issues the SMART team performed code analysis and architecture reconstruction.

3.4.2 Code Analysis

To address remaining issues, the team first analyzed the code through a code analyzer "*Understand for C++.*" This analysis provided

- a data dictionary
- metrics at the project, file, class, and function level
- an invocation tree
- a cross reference for include files, functions, classes/types, macros and objects
- unused functions and objects

The code analysis enabled the team to validate the input from the contractor and to produce input for the architecture reconstruction tool that would identify dependencies.

From the code analysis, it was found that the code was better organized and documented at the code level than most code the team was familiar with. However, as mentioned earlier, there were inconsistencies in the quality and documentation between different parts of the code that made the analysis complicated:

1. Since there was no consistent coding standard, individual differences between programmers could be identified. This made the code harder to understand.
2. Some parts of the code were difficult to navigate, with little cohesion and awkward file organization. Naming standards were different for files, classes, attributes, and method names. Code organization styles were different.
3. The organization of files was not standardized. For example, it was not clear why some files that did not perform user interface (UI) functions were located in UI directories. Another example is that some *include files* were co-located with code files and others located in a separate directory. Some files contained more than one class and there were no clear criteria for when this was allowed.

Despite these difficulties that forced the team members to become more familiar with the code than anticipated, they were able to produce the input for the architecture reconstruction tool.

3.4.3 Architecture Reconstruction

To address the issue of dependencies in more detail, the SMART team conducted an architecture reconstruction with a tool called *ARMIN*. Architecture reconstruction is the process by which the architecture of an implemented system is obtained from the existing system [Kazman 03, O'Brien 02].

To begin the architecture reconstruction, the team took the output from the code analysis and performed a focused analysis of the as-built architecture.

The team aggregated the code into several groups, each of which was dedicated to one of the following areas:

- component code that was identified as part of each service analyzed
- code directly dependent on the commercial mapping software
- user interface code
- the remainder of the code—data model, base classes, utilities, and code that did not belong to any of the above groups

In our analysis, the team was interested in dependencies between services and

- user interface classes
- the commercial mapping software
- other services
- the remainder of the code that mainly represented the data model

Through the analysis the team was able to identify a substantial number of undocumented dependencies between classes. These will enable a more realistic understanding of the scope of the migration effort.

The team was told that the architecture of the system followed the application of the *Model View Controller (MVC)* pattern. The architecture reconstruction found undocumented violations of the MVC architecture—specifically calls from the model to the view—that would need to be addressed in any migration effort.

The change from a standard system development effort to an SOA can have unanticipated impacts. For example, the product line approach used by the system developers was an excellent choice for the legacy application. However, the resulting architecture may increase the difficulty of the migration effort, since the large numbers of dependencies on core assets and the multiple levels of inheritance encoded may make it difficult to isolate stand-alone services needed for easy deployment in the target SOA. A solution to this problem might be to consider each service in itself as part of a product line, but this could require that the set of core assets be redefined.

3.5 Develop Migration Strategy

The recommended migration strategy can be summarized in the following steps:

1. Require the contractor to update the software architecture documentation and standardize comments in the code.
2. Work with the developers of the target architecture to define what is meant by a compliant service.
3. Work closely with the team within the target architecture group that is defining the data model to understand its contents and influence it as necessary.
4. Find out if the vendor has plans for a Linux version of the mapping software or if the target architecture group has plans for a mapping common service to replace the current Windows mapping software.
5. Interact with potential application developers that will be using the services to understand their requirements and develop appropriate service interfaces.
6. Recalculate cost and effort of migration based on a complete set of code dependencies and new understanding of user requirements and SOA constraints.
7. Understand the commonality between the current service migration effort and a second forthcoming similar migration project to a different target SOA.

In examining the potential for reuse of the existing legacy components, the team found that the current legacy code represents a set of components with significant reuse potential. However, because the current legacy system does not have sufficient architecture or other high-level documentation, it was difficult to understand the "big picture" as well as dependencies between classes.

To avoid this problem with future systems, the team recommended that the organization require the following changes from its contractors to make reuse of its legacy components more viable:

- documentation in the form of a suitable set of architectural views
- consistent use of programming standards
- documentation of code so that comments can be extracted using an automated tool
- documentation of dependencies, especially when they violate architectural patterns

A good starting point was provided by the analysis of the legacy components, based on the characteristics identified as important during the data-gathering activities. However, the team performed additional analysis of the code, as well as an architecture reconstruction to obtain additional data. The architecture reconstruction provided an "as-built" representation of the structure of the system and its dependencies. It suggested that the significant dependencies between classes will make reuse and deployment of services more difficult. If the migration effort moves forward, the results of the architecture reconstruction can be a starting point for understanding how to disentangle dependencies.

The largest risk in reusing the legacy components concerns the fact that the SOA has not been fully developed. While its overall structure has been defined, many of the specific mechanisms for interacting with it are still pending. Thus, it is not yet clear what the requirements for being a service in this environment will be in 12 or 18 months.

The impact that SOA decisions will have on the migration efforts is clearly seen in the concerns regarding the legacy data model. The architecture reconstruction allowed the team to document the central role of the data model, and to identify it as a potentially valuable reusable component, even though it had not been identified during the initial analysis. However, this finding was tempered by the fact that in the target SOA environment, potentially all services will have to use a common data model. If this is the case, all elements of the data model will require mapping to existing elements of the common data model. Negotiations will be necessary to make sure that all data elements needed by the services become part of the common data model.

To address the SOA instability issue head on, the team recommended that the organization take a proactive approach in working with the developers of the target SOA to understand the implications of the evolving SOA on services.

The organization should also work closely with the developers of the applications who will be using these services. Even though the technical part of the communication will be handled by a common service, the data to be transferred during that communication must be negotiated—the contents of both the request and the response message that is communicated between the application and the service must be defined. An initial and crucial element of discussion should be the data model, given that it is used by all the potential services.

Dependencies on the mapping software and other commercial products are a concern in the target environment. The Windows-based mapping software, for example, would need to be verified for use within the target SOA. A different mapping service might be required by the target SOA. There are also dependencies on a commercial database. These would have to be replaced by data access methods endorsed for the target SOA.

The team also noted that because there are dependencies between the primary services that were analyzed and a second forthcoming project that was being planned by the organization, there will be duplication of work if these are treated as separate projects.

4 Conclusions and Next Steps

The task of determining whether and how to expose legacy functionality as services can be complex. Disciplined analyses of existing components and the target SOA are necessary for sound migration decisions. SMART provides such disciplined analysis through a thorough and consistent process, a set of data-gathering activities that capture the scope of technical work to be accomplished, and artifacts that record critical aspects of the process.

We applied an early version of SMART to a command-and-control system and observed both significant potential for migration to services as well as shortcomings in documentation and code. In truth, the system owners will have a difficult time defining their services until the interfaces and expectations of the target SOA are better defined.

While the early version of SMART used to analyze the system proved valuable, there is significant room for improvement. SMART is being updated with the following goals in mind:

- Improve the breadth and consistency of information gathered about the engineering effort necessary to change the legacy artifact into a service. The SMIG is the first tool intended for this purpose. By incorporating significant technical "know how" into the SMIG, we also further an ultimate goal of transitioning the technique to other users.
- Incorporate decision rules on when it is most useful to include the code analysis and architecture reconstruction steps as part of the process.
- Develop machine support for capturing and analyzing data gathered during the SMART process. This will entail building templates for major artifacts, including the:
  - Stakeholder List
  - Characteristics List
  - Migration Issues List
  - Component Table
  - Service Table
  - SOA Description
  - Component Service Options Table
  - Migration Alternatives Table
  - Service Migration Strategy
  - Final Presentation
- Develop techniques and criteria for determining when a SMART team has captured sufficient information to complete the analysis process.

- Establish a mechanism to capture the net effect of SMART on migration efforts. This information is essential for continued evolution and improvement of SMART.

While SMART was designed with military migration efforts in mind, we believe that the technique has general applicability to organizations outside of the DoD. This is particularly the case when organizational goals involve more than just the wrapping of existing capabilities in order to make them accessible in an SOA.

As we continue to refine SMART, we plan to apply it to other projects and legacy systems. We are actively seeking organizations interested in applying the technique. We are also well on the way to establishing relationships with other organizations interested in adopting and improving SMART with us.

Appendix SMART Output

During the course of the SMART process, the lists and tables below are generated.

Stakeholder List: Created during "Establish Stakeholder Context" and updated during subsequent stages as necessary. The Stakeholder List identifies stakeholders and the type of information to be elicited from each, and provides contact information.

Characteristics List: Created during "Establish Stakeholder Context" and updated during subsequent stages as necessary. The Characteristics List identifies information about components to be gathered and later considered in determining whether service migration is feasible and appropriate. The Characteristics List is composed of a set of predefined characteristics that have been developed based on SEI experience, combined with the additional characteristics identified during the SMART process.

Migration Issues List: Created during "Establish Stakeholder Context" and updated during subsequent stages as necessary. The Migration Issues List identifies concerns that must be addressed during the migration process. Some migration issues may be applicable to all components and services (i.e., general issues), while others may be applicable only to specific components or services.

Component Table: Created during "Describe Existing Capability" and updated during subsequent stages as necessary. The Component Table identifies components under consideration for migration to service and is used to capture characteristics (identified in the Characteristics List) of each component under consideration.

Service Table: Created during "Describe Existing Capability" and updated during subsequent stages as necessary. The Service Table identifies potential services that can be derived from components and captures information regarding these potential services.

Component Service Options Table: Created during "Analyze the Gap" and updated during subsequent stages as necessary. The Component Service Options Table identifies each potential mapping of legacy components (Component Table) to potential service (Service Table). Mapping need not be one-to-one. A component may have to be split into several services, or several components may be combined to form a service. Also, several ways of producing a service may be possible. This may involve alternate ways of using available components, as well as using capabilities that were not part of the original system, such as COTS products. The Component Service Options Table is updated with information about cost, effort, and risk for each option during "Develop Strategy for Service Migration."

Migration Alternatives Table: Created during "Develop Migration Strategy." For each recommendation in the Component Service Options Table, there may be more than one viable strategy to achieve the migration goal. These strategies may vary along many dimensions, such as the components selected for migration, the sequencing of migration activities, the use of external services, and the types of modifications made to the code. Viable strategies are documented as entries in the Migration Alternatives Table.

Service Migration Strategy: Developed during "Develop Migration Strategy." The Service Migration Strategy summarizes the SMART process undertaken and information gathered, and provides a preliminary analysis of the viability of migrating legacy components, to services, migration strategies available, and the costs and risks involved. In particular, it attempts to answer several questions:

1. Which components can reasonably be used to derive what services?
2. What sorts of activities must be performed to accomplish the migration?
3. What strategies are most appropriate for the migration effort?

Final Presentation: A slide (plus notes) presentation developed during "Develop Migration Strategy" that summarizes the process and migration strategy and is presented to the customer.

References

URLs are valid as of the publication date of this document.

[Alberts 00]	Alberts, D.; Garstka, J.; & Stein, F. *Network Centric Warfare: Developing and Leveraging Information Superiority*—2nd Edition (Revised). Arlington, VA: CCRP Publication Series, 2000.
[Bergey 02]	Bergey, J.; O'Brien, L.; & Smith, D. "Using the Options Analysis for Reengineering (OAR) Method for Mining Components for a Product Line," 316-327. *Software Product Lines: Proceedings of the Second Software Product Line Conference (SPLC2).* San Diego, CA, August 19-22, 2002. Berlin, Germany: Springer, 2002.
[Brown 02]	Brown, A; Johnston, S.; & Kelly, K. *Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications*. Santa Clara, CA: Rational Software Corporation, 2002.
[Kazman 03]	Kazman, R; O'Brien, L.; & Verhoef, C. *Architecture Reconstruction Guidelines,* 2nd Edition (CMU/SEI-2002-TR-034 ADA421612). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents /02.reports/02tr034.html.
[Lewis 05]	Lewis, Grace & Wrage, Lutz. *Approaches to Constructive Interoperability* (CMU/SEI-2004-TR-020). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/04.reports/04tr020. html.
[O'Brien 02]	O'Brien, L.; Stoermer, C; & Verhoef, C. *Software Architecture Reconstruction: Practice Needs and Current Approaches* (CMU/SEI-2002-TR-024, ADA407795). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. http://www.sei.cmu.edu/publications/documents/02.reports /02tr024.html.

[Potter 05] Potter, John. *Statement of Postmaster General /CEO John E. Potter.* United States Postal Service, 2005. http://www.usps.com /communications/news/speeches/2005/sp05_0426pmg.htm.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 2005	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE SMART: The Service-Oriented Migration and Reuse Technique		5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) Grace Lewis, Ed Morris, Liam O'Brien, Dennis Smith, Lutz Wrage		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2005-TN-029
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

This report describes the Service-Oriented Migration and Reuse Technique (SMART). SMART is a technique that helps organizations analyze legacy systems to determine whether their functionality, or subsets of it, can be reasonably exposed as services in a Service-Oriented Architecture (SOA). Converting legacy components to services allows systems to remain largely unchanged while exposing functionality to a large number of clients through well-defined service interfaces. The U.S. Department of Defense (DoD) is adopting this approach by defining SOAs that include a set of infrastructure common services on which organizations can build additional domain services or applications. SMART considers the specific interactions that will be required by the target SOA and any changes that must be made to the legacy components. An early version of SMART was applied with good success to assist a DoD organization in evaluating the potential for converting components of an existing system into services that would run in a new and tightly constrained DoD SOA environment.

14. SUBJECT TERMS Service-Oriented Migration and Reuse Technique, SMART, service-oriented architecture, SOA, migration, legacy components			15. NUMBER OF PAGES 39
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL