Chapter 5
The Fractal Nature of SOA Federations: A Real World Example

Arthur Baskin, Robert Reinke and John W. Coffey

Abstract Fractal concepts are often said to be recursively self-similar across multiple levels of abstraction. In this paper, we describe our experience with the fractal nature of SOA designs for sustainment management tools as these tools evolve into even more dynamic, federated systems that are orchestrated over the web. This chapter summarizes insights gained from more than twenty years of software development, maintenance, and evolution of a major pavement engineering tool named PAVER™. We consider both theoretical and experiential aspects of SOA federations at three levels of abstraction: (1) a loosely coupled federation of enterprise systems with PAVER™ as one member, (2) a tightly coupled federation of two pavement management tools (PAVER™ and PCASE) where each has a separate domain identity and development team, and (3) an emerging federation of plugin tools, which provide additional pavement engineering functionality and can come from competing civil engineering firms. These plugin tools exist at different levels of abstraction within the level of the main system and are, again, fractal. We organize the presentation of our experiences in this domain by describing how SOA elements including Ontologies, Discovery, Composition, and Orchestration are fractal whether we are looking at algorithms or persistent state. We also define and describe a third orthogonal fractal dimension: Evolution. Although the details of the implementation solutions at the differing levels of abstraction can be substantially different, we will show that the underlying principles are strikingly similar in what problems they need to solve and how they generally go about solving them.

A. Baskin (✉) · R. Reinke
Intelligent Information Technologies Corporation, Indianapolis, IN 46216, USA
e-mail: abaskin@intelligent-it.com

J.W. Coffey
Department of Computer Science, The University of West Florida,
Pensacola, FL 32514, USA

E. El-Sheikh et al. (eds.), *Emerging Trends in the Evolution of Service-Oriented and Enterprise Architectures*, Intelligent Systems Reference Library 111,
DOI 10.1007/978-3-319-40564-3_5

5.1 Introduction

It is the goal of this article to make the case that fundamental concerns in the development and evolution of SOA systems are self-similar or fractal across different levels of abstraction in SOA systems. These observations about the fractal nature of federations of service oriented systems (SOA) are grounded in more than twenty years of software development and maintenance of a series of condition-based civil engineering maintenance management systems. Foremost among these systems has been the PAVER™ system, which is a pavement management system for airfields and roadways. In some cases, these decision support tools are targeted at elevating the expertise of a normal practitioner to be closer to that of the expert, whose expertise is embodied in the system. In other cases, these systems support a loose federation of human decision makers, who are again engaged in complex decision making. For example, the PAVER™ pavement management system embodies international standards for ways to convert visual observations of the pavement into a standard pavement condition index, which can be used prioritize the use of scarce resources for civil infrastructure repairs.

The author's experiences with many of the principles, which are described here, predate the general emergence of SOA concepts, and, therefore, we interpret some of the historical material looking backward. To enable this perspective, we provide a brief historical account of PAVER™. We justify this retrospective interpretation of history because we were responding to the needs for modular evolution of separate tools and the need for loose coupling among tools to allow human decision makers to compose data from varying sources to make engineering judgements. We assert that these are some of the same requirements that SOA systems are proposed to address. Increasingly over the last ten years, we have been actively injecting insights from the SOA approach into the structure and organization of our software systems. In fact, the preparation of the material for this chapter has had a direct impact on the emerging support for plugins to the pavement engineering desktop because we have used SOA principles to guide the definition and implementation of the plugin framework, which has overlapped the development of the material for this chapter in time.

In what follows, we try to present both the experiences that drove us to arrive at or validate principles as well as a theoretical basis for the principles where we believe we have found one. Two of the authors (Baskin and Reinke) have been involved with software development tasks where attention to the evolution of the software was part of the problem and the time to reflect on why some development techniques work better than others was available. In some situations, the principles emerged from the experience supporting the evolution of the software and in other situations, we were able to bring principles from other disciplines to bear on our software develop projects.

We have attempted to organize what we have learned at two distinct levels of abstraction: (1) overall principles and (2) origin of the principles. Where possible, we have attempted to provide both a theoretical derivation of the principles and

concrete examples of the principles, which are drawn from our experience with pavement management systems. Readers may wish to pursue either the theoretical or historical description of the principles or simply review the summaries of what we believe we have learned.

We have found SOA Federations to be effective for managing complex software systems: Across time with the interplay of (relatively) independent actors, domains of problem solving, and goals (with or without a common owner). Unpacking this deceptively simple statement is our goal in this chapter. Because many of the terms we use do not have precise definitions (or even agreed definitions), we will spend some time in each section describing what we mean by our terms. These definitions will be especially important because we need to use them across several fractal levels of abstraction. Although we focus on the domain of pavement engineering maintenance management systems in this chapter, we have applied these principles in the development of decision support systems in other areas of civil engineering and in shortening time to market for new products by using agile software development techniques to begin to develop software to test a new product even before the design of the new product has been finalized.

Figure 5.1 shows the three fundamental dimensions of our separation of issues for SOA Federations.

In the Fig. 5.1, we identify traditional SOA issues along one axis. At the risk of oversimplification, we can say that ontologies precede and are necessary for supporting Discovery, Orchestration, and Composition. In our analysis, we look at ontologies for procedural and declarative aspects of a problem and the need for evolution of ontologies, which might be the hardest problem of all to solve. The remaining three SOA issues: Discovery, Orchestration, and Composition can be thought of as describing the state of a particular SOA system at one point in time. That is, a working SOA system must discover procedural and declarative elements so that they can be composed and finally orchestrated. In our SOA desktop for pavement engineering, the desktop follows a discovery process each time it is started to determine the components, which are available for composition in this user session. The human user plays a central role in orchestrating the interoperation of some of the components and simplifies the software orchestration problem for us.

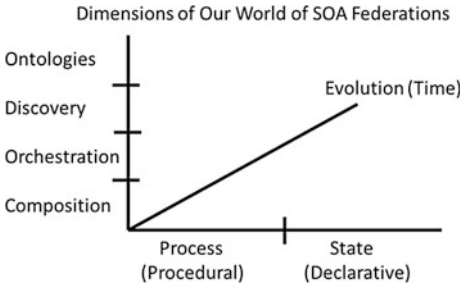

Fig. 5.1 Three dimensions of issues for fractal analysis. We separate SOA issues into procedural and declarative approaches to software construction. Finally, we explore the evolution of all of these aspects over time

5.2 The Historical Context of This Work

Our work on PAVER™ started with a series of workshops, which were sponsored by the Federal Highway Administration and supervised by Shahin [1] who continues to guide these efforts for the U.S. Army Corps of Engineers, which owns the software. The workshops brought together researchers in pavement engineering, practicing pavement engineers, and our software development team. These workshops guided the development of the software to help define the best practices in the pavement management domain and to embody them in a revised version of the PAVER™ system. These user group meetings continued for years after the startup phase ended and the interplay of the expert users and the software designers has supported the co-evolution of best practices for pavement management and the design of new PAVER™ features [2]. New features have also been driven by pavement engineering research carried out by Dr. Shahin and others. This emergent requirement for software evolution as a consequence of best practice evolution, which was, in turn, partially stimulated by the PAVER™ software system, gives rise to our attention to evolution as a basic area of interest.

The PCASE system evolved largely independent of PAVER™ but was used by many of the same pavement engineers at the same locations. About fifteen years ago, the U.S. Air Force funded an integration program to bring PAVER™ and PCASE together in what the user would experience as a seamlessly integrated system. In addition, each separate system had to be able to be deployed independently as it had been done in the past. A user needed to be able to install either of these civil engineering tools first and then optionally add the other. Finally, this federation of loosely coupled tools had to be developed by two different research groups with two largely disjoint user groups and rates of evolution. Our experience with such a federation of largely independent software development efforts that produce a seamless integration of the tools in the user experience has led to our attention to the issues of discovery, orchestration, and, again, evolution of software systems in federations of civil engineering tools.

In the latest iteration of work on these pavement engineering tools, we have been tasked by the Corps of Engineers to expand the existing tools to provide support for international users, interoperation with other Department of Defense enterprise systems, and to support the incremental evolution of these pavement engineering tools through an enhanced version of the plugin mechanisms, which have been available but underutilized for ten years. We have explicitly used the principles described in this chapter to guide the additions to the plugin machinery and to suggest requirements that might not otherwise have emerged from the domain requirements.

5.3 Literature on SOA Federations, SOA Elements, Algorithms and Data Persistence

In the following section, we provide a brief overview of literature pertaining to SOA federations, the SOA elements we consider to be fractal in nature, and the relationship between the cross-cutting concerns of algorithms and persistent data. An extensive literature is available on the topic of SOA federations. Zdun [3] describes a federation as a controlled environment with a collection of peer services. Each service might be controlled by one or more federations, but within a single federation, peer services can easily interact. Federations are based upon middleware that fosters interoperability of loosely-coupled services. The term federation appears in the literature in two contexts and it has evolved out of two distinct but increasingly interrelated technological bases. The High-Level Architecture, and accompanying Real-Time Interface (HLA-RTI) [3–5] originated in the mid-1990s in the DoD as a platform for real-time simulation. The IEEE 1516 standard grew out of the High-Level Architecture initiative. In this context, a federation is a collection of simulation entities connected and orchestrated by the real-time interface. A federate is an individual IEEE 1516-compliant simulation entity. The HLA includes a federation management API [6].

By the late 1990s, well after early versions of PAVER™ were already in use, SOA was an emergent technology for the creation of composite applications. SOA is based upon W3C standards including WSDL, SOAP, and XML Schema [7–9]. The HLA-RTI notion of federation is somewhat distinct from the more general notion of federation as described in SOA literature because it is a particular use in simulations rather than for the creation of composite applications that accomplish a broad range of business or engineering purposes. However in more recent times, calls for integration of HLA and SOA have occurred [5] as it is viewed that lessons learned in one community might benefit the other. Additionally, benefits might be obtained by the interoperation of both standards. For instance, Seo and Zeigler [10] described the DEVS/SOA system to provide web service-based simulations.

A major concern in the creation of federations involving multiple providers is managing identities and access of federates [11]. Several standards-based protocols have been proposed or implemented to create federations [12–14]. Li and Karp claim that the federated identity management approach has proven to be difficult to use and upgrade, and is not scalable. They state that federation based upon identity is the wrong focus, rather the focus should be on access management instead. They illustrate implementing access control policies using SAML certificates [15]. Thomas and Meinel [16] describe their own management system, which also relies on open web service standards, to provide reliable digital identities. Hatakeyama [17] describes what is termed a “federation bridge” to facilitate cross domain identity federation. Anastasi et al. [18] state that service providers offer their services using proprietary management software, interfaces and virtualization technologies which make interoperability more difficult to achieve. They discuss their simulation tool SmartFed which is designed to simulate cloud-based federations.

Since a detailed review of literature pertaining to the SOA elements of discovery, composition, orchestration and ontology would encompass an entire book chapter in itself, we refer the interested reader to the following representative works. Al-Masri and Mahmoud [19] attempt to incorporate client goals into service discovery queries via a means that would rank candidate services in a manner similar to query result rankings implemented in general-purpose web search capabilities. Dsbrowski and Pacyna [20] address inter-domain service discovery and claim that service discovery systems will require a strongly interwoven identity management component. They state that support for service discovery across service domain boundaries must be implemented in identity management systems in order to provide a safe discovery system between services from different business areas.

Tolk et al. [21, 22] have done important work on composition and orchestration within large governmental SOA federations. They describe current Homeland Security systems which must integrate data and processing capabilities from twenty two previously separate agencies. In [21] they describe model-based, top-down orchestration of heterogeneous Homeland Security systems with discovery and composition of IT capabilities included in a system-of-systems bottom up. In [22] they describe a mathematical model for the selection or elimination of candidate services, and for their orchestration and execution. They describe this work as a first step towards self-organizing federation languages. Work by Rathnam and Paredis [23] provides an ontology-based framework to simplify the reuse of federates in a federation object model. While their work pertains to HLA-RTI federations, it is applicable in principle to SOA systems in general.

We have identified as key issues in the current work, the cross-cutting concerns of algorithms and data persistence in SOA. Calvanese et al. [24] discuss this issue stating that one's view of the pre-eminence of data or process is often a function of one's viewpoint, for instance if one is a business process analyst or a data manager. They cite an article by Reichert [25] which makes the claim that "data and processes should be considered as two sides of the same coin." They discuss "data-aware processes" and conclude that the database theory community has developed the defining techniques to deal with data and processes. They cite several important issues including verification issues pertaining to the modeling of what data can be changed by a process. Dobos et al. [26] describe a platform for the management of reusable process components and for the federation of data stores in order to support data persistence, statistical analysis and presentation of the data.

Data persistence is an important aspect of data management. Krizevnik and Juric [27] cite data persistence problems stemming from poor data quality, heterogeneity of data sources and poor system performance in SOA systems. They describe a SOA persistence model relying on master data management (MDM), and data transfer standardization by the use of service data objects (SDOs) [28] in order to build a data services layer in a SOA system. Software companies seek to build in data services layers in their SOA architecture solutions. For instance BEA Systems [29] describe the AquaLogic Data Services Platform (ALDSP) stating that the environment employs a declarative approach to the construction of data services

that are based upon XML Schema for data definitions and XQuery as the service composition language.

Takatsuka et al. [30] state that cloud computing and machine-to-machine technologies require "context-aware" services to deal with heterogeneous data from distributed systems. They describe a rule-based framework to create context-aware services where context is taken to mean situational information that can be true or false. Sarelo [31] describes the HERMES framework for ubiquitous communication management using web services with serialized XML, data replication with peers storing full copies, and simultaneous data update of all replicates. The previous literature review barely scratches the surface of available literature on all these aspects of SOA, but it provides the interested reader with a starting point for further exploration.

5.4 Three Levels of Abstraction for SOA Federations

Although the development of the systems of interest in the current work has evolved from support tools for a single practicing field engineer toward the needs of enterprise systems, we may now look at these emerging systems from the top down. Figure 5.2 shows the three major levels of abstraction for the tools which we discuss in this chapter:

Dr. Shahin's work on condition-based maintenance management systems gave rise to a series of similar system. We put forth RAILER, which is a system for maintenance management of railroad track, and ROOFER, which is a system for maintenance for roofing on buildings, as examples of this more extended family of civil engineering systems. Taken together, these systems, which are focused on largely disjoint assets, form an enterprise level system for the application of best

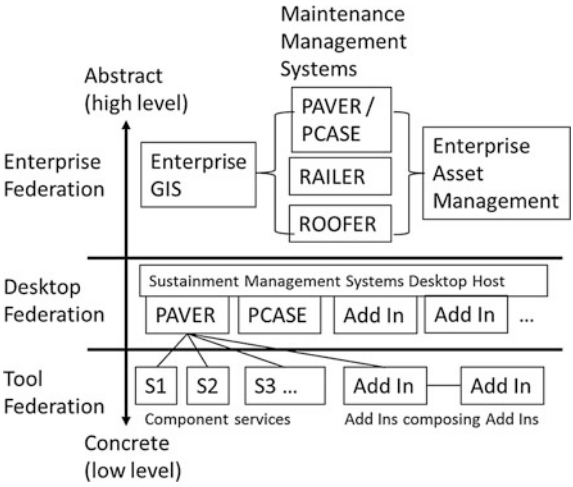

Fig. 5.2 Three levels of abstraction for SOA Federations: (1) Enterprise Federation, which brings together separate line of business systems for exchange of summary data; (2) Desktop Federation, which brings together more tightly coupled systems with separate identities and intersecting user functionality, (3) Tool Federations, which break individual desktop tools into component services as a technique for managing complexity or ease of extension

practices for maintenance management so as to provide the safest and most productive use of the civil infrastructure assets for the lowest possible cost. There is increasing demand for interoperability among these separate systems with separately developed enterprise systems for geographical information systems (GIS) and enterprise systems for real property and asset management. GIS systems are used to integrate disparate data bringing together data from separate systems and overlaying these data geospatially. Asset management systems are used to track the value of assets and to plan for allocation and preservation of these assets.

The development of the desktop federation of PAVER™ and PCASE was driven by the user need for these systems to share a common inventory and constrained by the need to allow the tools to retain their separate identities while being able to be seamlessly combined in a single desktop when needed. One of the greatest challenges in this federation was to support the separate rates of evolution of PAVER™ and PCASE and to accommodate the various possible combinations of separate versions.

In its latest incarnation, PAVER™ has been modified to support multiple users and multiple deployment options, which include the traditional standalone install, a thick client-server installation, a shared remote application server, and a web browser interface. As part of managing both the increased pavement engineering capabilities and the varied deployment options, we have used traditional SOA concepts to break the main PAVER™ application into a family of interoperating component services, which can be dynamically loaded in a standalone application or accessed over a web service interface to a remote server. The user is able to switch seamlessly between these modes on each "File/Open" operation, which might open a local database with local services one time and a remote database with remote services the next. This Tool Federation has been extended to support plugins (called Add Ins in PAVER™). Plugins can be a way to extend the core pavement engineering modules and a way to bring together tools from other pavement engineering companies to leverage the common inventory and GIS display tools in much the same what that PCASE can do.

5.5 Dimensions of Our SOA World at Each Level of Abstraction: Real World Example

This section describes our experience with each of the elements in our three dimensional space at each level of abstraction as an extended real world example of these issues at three distinct levels of abstraction. The next section identifies the fractal issues, which have been found to be common to these different levels of abstraction.

5.5.1 Enterprise Federation

The Enterprise Federation of relatively independent systems is a major initiative in the U. S. Department of Defense (DoD) and the work on pavement management systems has been an early emphasis. Unlike the more mature developments at lower levels of abstraction, the Enterprise Federation is still a work in progress; therefore, the results of our analysis at this level are a bit more tentative but we are able to bring lessons learned from the lower levels of abstraction to bear at this level to help guide the software development.

5.5.1.1 Ontologies

Declarative: The PAVER™ system is considered the authority of record for condition-based maintenance of roadway and airfield pavements in the U.S. DoD and for NATO. Using the system, a pavement engineer obtains data about the state of the pavement, its likely future condition, and the cost of foreseeable maintenance. Because the pavement engineer will usually actually stand on or drive over the pavement to make an inventory and condition assessment, the pavement engineer can determine the specific pavement quality and present usage. In addition, the pavement engineer will break the overall pavement extent into manageable sizes for future work planning. The enterprise GIS system and PAVER™ must share a common geospatial rendering of the overall pavement extents (a shared ontology) and PAVER™ can be used to subdivide the overall area into subcomponents, which are the smallest unit of measure on the pavements (called Sections). Sections, again, form a shared ontology between the systems. In this case, the GIS system is the authority of record for the overall pavement extent and PAVER™ is the authority for the section boundaries, which are needed for work planning. In a similar way, the pavement engineer is often in the best place to tag the pavement according to the required asset management attributes, which are called CATCD (category code to summarize type, use, and cost) and RPUID (Facility identifier). Within this rigid standardization, a facility can be divided into "segments" as long as each segment is part of the facility and can be assigned a CATCD. In this second case, the shared ontology is closely controlled by the real property system (CATCD and RPUID) but the segmentation and the assignment of attribute tags to elements of pavement might be shared between the asset management system, which is the authority of record, and the pavement management system because the pavement engineer is more likely to actually stand on the pavement. Definition of these shared ontologies has consumed many hundreds of hours of group meetings, which have involved many more systems than PAVER™, and the implementation of the software support for the emerging ontology has been the simplest part.

Procedural: Unlike the shared declarative ontology, which can be shared between the U.S. Air Force, Navy, and Army, the procedural ontology varies by service because the GIS and real property asset systems are not the same. The state

of the procedural ontology for reconciling data at the enterprise level is at the same place that PAVER was more than twenty years ago. There have been many meetings and deliberations collecting required data, reconciling the differences between the mandated authoritative sources for the data and the operational sources for the data. The PAVER system is only one of sixteen major systems participating in this process. The fact that pavement management data must interoperate with different enterprise level systems and be collected according to a variety of business rules means that the ontology must be able to evolve with the emerging changes. As the enterprise data become more available, the best practices as the enterprise level will co-evolve alongside the software systems in the same way that best practices for pavement management and PAVER co-evolved twenty years ago.

Evolution: The enterprise federation brings together different types of structures: vertical (e.g. buildings) and horizontal (e.g. roadways, power grids) for a variety of largely independent entities (e.g. Air Force, Navy, Army, municipalities). The real property/asset management ontology relates physical structures to congressional authorization for funding and is closely controlled. The individual services have evolved separate approaches for the segmentation of assets and the assignment of CATCDs. In the process of integrating sixteen different civil engineering disciplines, of which PAVER™ is just one, the formal ontology for sharing of GIS data has gone through several major versions and version four of the specification is nearing completion and has shown that the services must support evolution of their components driven by their history, current needs, and future missions.

5.5.1.2 Discovery

Declarative: The discovery problem for the enterprise federation is still being solved by ongoing business process reengineering and software development. Each service is developing its own standard operating procedures for how these data should be collected, coordinated, and used for resource allocation and sustainment of the assets. These declarative procedures are tailored to the details of the operations of each service.

Procedural: As a practical matter, these systems are all composed of a union of humans and software systems. In some situations, procedures can be automated and in others, some form of engineering judgement about the interpretation of the data is required. Some agencies have chosen a focus on data replication and some on direct linkage of data. In some situations, the process can be fully automated and in others only semi-automated. The GIS systems and the real property linkage must be able to tolerate the variety of presentations of shared data from each service and/or from each civil engineering domain.

Evolution: The methods for discovery must be able to vary across the services (a form of evolution across situations rather than time) and they must be able to evolve along with changes in technology and best practices. The methods for this type of modular replacement of parts are still being developed, but the problem is clearly

present in the variety of approaches to this data alignment problem continues to increase.

5.5.1.3 Composition

Declarative: The PAVER™ inventory has long been "composed" of Networks, which contain Branches, which, in turn, contain Sections. A Section is the smallest unit of maintenance activity and must have both a uniform structure and history. A section may be broken into samples, which allows quicker inspections by extrapolating data from samples to the entire section instead of inspecting everything. The GIS data integration problem is being addressed by the notion of a map which is "composed" of layers. Each layer can be associated with an engineering discipline, e.g. pavements, and data attributes for a region in that layer can come from either geospatial data (e.g. area), asset management (e.g. CATCD), or a maintenance management system (e.g. condition of the asset). A maintenance system is usually expected to rollup lower level data (from segments) into values at the facility level.

Procedural: The procedural aspects of composition at the enterprise level are easiest to see in the GIS presentations, which are data visualization tools at their core. Much of the composition of maps can be interactively driven by a human viewer, who can turn layers on and off as well as selecting among a variety of coloring strategies. In the pavement domain, image capture devices are now able to capture roadway images, which can be analyzed for distresses and dynamically aggregated into "samples," which can then treated like the more traditional samples at the bottom of the pavement composition structure. Rollup of data over the composition structure can be seen as flattening the depth of the composition to get summary data and the flattening algorithms can be weighted by size or importance to mission.

Evolution: The composition of the various engineering maintenance management systems is evolving as some new condition-based maintenance management systems are added and others consolidated. The composition of GIS information in the form of layers and coloring strategies ("themes") is constantly changing as new theme definitions are defined and new attributes added to each layer.

5.5.1.4 Orchestration

Declarative: The prevalence of standard operating procedures for components of the Enterprise Federation provides a roadmap for orchestrating the interaction of the systems, which is frequently driven by human users. These best practices for orchestrating the interaction of the data and systems are still being developed.

Procedural: As much as practical, the orchestration of the interaction among these systems will be automated. Again, because this level of abstractions is the newest, the development of orchestrations algorithms is in its infancy.

Evolution: The orchestration of the interaction among the members of the Enterprise Federation is evolving as the different historical contingencies of the various services (Air Force, Army, and Navy) are incorporated and as the differing enterprise systems for GIS and real property asset management are included. The orchestration techniques must be allowed to evolve with some independence as each service meets its different mission needs.

5.5.2 Desktop Federation

The desktop federation of PAVER and PCASE has been hosted on a common pavement engineering desktop for the past fifteen years. Users can install either program alone or install both to get a seamless integration of the tools.

5.5.2.1 Ontologies

Declarative: The integration effort began with many months of matching up key concepts in the systems in order to arrive at a shared ontology for the shared inventory elements. The shared ontology contained Network, Branch, and Section from PAVER as well as the pavement use and surface type. The notions of non-destructive test data and layer definition were taken from PCASE. Additional ontological elements were identified as predominantly being associated with one system but these concepts are useful, in principle, to both.

Procedural: Each of the two members of this federation had its own database structures behind the ontologies. Once the competing ontologies were reconciled, we needed to unify the persistence while respecting the constraint of allowing each system to retain its separate identity and independent development team. We accomplished this by having the system construct a single logically unified database by linking the various databases from PAVER and PCASE into it. In this way, we could manage a unified common set of core inventory data and allow for the union of additional persistent data, which is managed by one member of the federation but could be reported by either.

Evolution: The PAVER and PCASE teams have been free to control the portion of the ontology, which is predominantly controlled by one group. Changes to the core shared inventory elements required both groups. At one point, PAVER had two different versions (versions 5 and 6) deployed with different file formats and internal object structures. Backward compatibility with PCASE was provided by use of a façade, which made version 6 appear to provide the same ontological model as version 5 to PCASE.

5.5.2.2 Discovery

Declarative: The desktop federation of PAVER and PCASE might be comprised of either application alone or both together. We honored this constraint by having the Desktop executable search for a file with a special extension so as to discover what tools were available. These files described the root level application object to instantiate and put into a collection of application objects being hosted by the desktop. In PAVER versions 5 and 6 and PCASE versions 2.08 and 2.09, the menu system for each application was also declaratively stated in a tabular format and discovered by the desktop load on startup.

Procedural: PAVER versions 5, 6, and 7 all use a search to discover applications to load. Starting with Version 7, the menu system has been described procedurally and merged on desktop startup. The Version 7 desktop can also be used to host additional tools, which are derived from PAVER, i.e. the Image Inspector (for analyzing roadway images) and the Field Inspector (for collecting distress data in the field) both use this same desktop together with a procedural menu system and a configuration file to be discovered by the desktop when it loads.

Evolution: For the past fifteen years, the discovery mechanism in each version of the shared pavement management desktop has supported the independent evolution of PAVER and PCASE and, more recently, plugins.

5.5.2.3 Composition

Declarative: The shared ontology dictates that both PAVER and PCASE have a shared inventory, which is a composition of inventory levels. The shared ontology also defines time series data, which are items with a date under the sections, e.g. Work History, Inspections, and Conditions.

Procedural: In addition to the declarative (predefined) compositions, users and tools can add members to the collection of "condition measures." PCASE adds measures and users may add their own measure scales. When additional condition measures are added, it is as if there are now more compositions of conditions available for data entry and reporting. The GIS reporting tools and the tabular reporting tools will detect the new compositions of condition data and show the user reports with the new types of data.

Evolution: The available condition measures have evolved with the evolution of best practices and the advent of more automated data collection tools. In addition, users can control the presence of certain compositions of descriptive fields, which are used for asset management and even hide some of the compositions or repurpose them for another use in the system. In this way, the system supports a limited amount of evolution of the composition of the data. The advent of "Add In" modules in version 5, which have been expanded in version 7.1, brought the ability to add procedural behavior to these new compositions of data and to compose behaviors for new compositions through procedures encoded in a dynamically loaded plugin module.

5.5.2.4 Orchestration

Declarative: We say that the Desktop "orchestrates" the separate application objects (PAVER and PCASE) because it handles the process of wiring up the objects and passing action messages from menu items and such to the correct application object. This declarative definition of the orchestration behavior is enforced by interface contracts, which an application object must honor. Starting in version 7, the declarative machinery was enhanced to allow application objects and individual component tools within an application object, to register "interest" in one or more events and to be notified when they occur.

Procedural: The shared desktop orchestrates some activities with a combination of procedural processes and notifications, e.g. when a user changes unit system (e.g. Metric to English) or changes language (e.g. from English to Italian), the desktop can handle some of these operations itself and must orchestrate the notification and update process for all of the tools within all the applications and plugins. In versions 5 and 6, notifications were broadcast to all participant tool objects independent of declared interest in the changes. In version 7, tool objects can register to receive notifications and are responsible for handling them without further help from the desktop. This change of architecture was needed to allow the same mechanism to be used for Windows forms and web pages, which realize their user interfaces in radically different ways but can share this orchestration logic.

Evolution: As mentioned before, the desktop orchestration machinery has been required to evolve in order to handle PCASE 2.08 and PCASE 2.09 together with versions 5 and 6 of PAVER. More recently, the same family of orchestration tools has been used to control a version of PAVER 7 for Windows and the Web. Finally, the orchestration machinery for notifications about a user's focus of interest is open ended and can be extended by tools at runtime to support notification messages, which were not originally predefined.

5.5.3 Tool Federation

The Tool Federation represents a subdivision of the PAVER level of abstraction into a collection of partially separable tools. In version 7, the system was extended to support multiple users in either a web or thick client—server configuration. The client-server interaction is through web services using Microsoft's Windows Communication Framework (WCF). Unlike many stateless web service protocols, WCF supports the notion of a user session and we have extended that notion to allow us to have web services, which can initialize web services with a user specific data context and return what amounts to a handle to the web service.

Web service protocols are often stateless and each service call must stand on its own. In our systems, we often need to have what amounts to a web service that can return a handle to another service where the new service shares common user data. We use the WCF notion of a user session to implement what amounts to web

services that can return web service handles in order to support complex chains of processes. In addition to the separate web service tools within PAVER, the Tool Federation also includes the "Add Ins" (plugins) to the desktop, which can also extend PAVER. These additional modules live almost entirely on the client, but they have access to the persistent data on the server through a constrained interface. Because these tools live within the level of the overall PAVER system, they have access to many of the SOA elements of the overall application. In this section, we discuss only those aspects that are specific to the Tool Federation.

5.5.3.1 Ontologies

Declarative: The WCF protocol makes the distinction between declarative ontologies (Data Contracts) and Procedural Ontologies (Operation Contracts). These data contracts are not interesting for this analysis because they are more an artifact of technology than a true ontology. Our preference for developing models of the application domain and then solving problems using the model makes these data contracts more like ontologies than they would otherwise be. The declarative ontologies for plugins are directly related to this material. Version 71, extends the machinery for plugins to be able to store data in the main inventory database as well as the separate and potentially shared system tables database. These data can participate in the Import/Export process for sharing data between users by file sharing and updates to these data are multiuser safe. The Desktop imposes an overall declarative structure on these data, which involves a GUID for the plugin and a "Class Name," which must be supplied by the plugin when retrieving or storing data. The "Class Name" is basically like the name of a component ontology, which is managed by the plugin. This block of data can consist of a miniature database, which is composed of a collection of potentially interrelated tables. The plugin provides the meaning for the internals of these tables and the Desktop imposes the structure of the key fields used to distinguish the individual blocks of data.

Procedural: Again, the procedural ontologies associated with the WCF services are needed in order to break PAVER into a set of component web services, but they are relatively uninteresting for this work. The plugin machinery provides all of the aspects used by WCF and several more. The procedural ontologies are defined by a special interface contract DLL component to which plugins must conform. The various interface contracts define how the desktop and the plugins can communicate. For example, a plugin can place menu items at various locations in the system (main menu items or GIS map display options) and the procedural ontology for plugins governs how this is done. In addition, the plugin itself controls the semantic meaning of the items and their associations with other items through its algorithms for data collection, validation, value added processing, and presentation.

Evolution: The plugin machinery provides a means for ontological evolution by its mere presence. The high level constraints of the storage of mini-databases only modestly constrain how these items are packaged. In some situation, such as for

non-destructive testing, there are domain ontologies, which can be supported by plugins even when these ontologies vary from one equipment vendor to another.

5.5.3.2 Discovery

Declarative: The Desktop uses declarative information about an inventory, which the user can access through File/Open, to determine whether to access data locally or on a remote server. The discovery process for plugins is more interesting for this analysis. Each user has a declarative (tabular) set of preferences, which specifies whether an available plugin should be loaded for that user or not. The desktop manages the discovery of installed plugin modules on startup and handles the dialog with the user to discover which items the user wants to access. The selection of plugins to use is persistent between user sessions and is user specific.

Procedural: The Desktop uses a procedural search together with requirements for contracts to be implemented by plugins to discover available plugins on startup. For software security reasons, this discovery process is constrained to distinguish between "certified" plugins and "external" plugins and is constrained to a tightly controlled set of locations. The underlying procedural discovery and composition of plugins is capable of supporting less restrictive policies. Perhaps the most interesting discovery operation involves plugins being able to discover other plugins. When a plugin is loaded into the collection of active plugins, the desktop can determine if the plugin is willing to be visible to other plugins. A plugin with a dependency on another plugin can determine if the other plugin has been loaded and can obtain a pointer to the other plugin. After the desktop has facilitated the connection between the plugins, they may continue to cooperate directly or indirectly through the database and/or the user interface.

Evolution: The plugin machinery exists for the purpose of allowing the discovery of incrementally added modules of functionality. Thus, its primary purpose is to allow the main desktop system to evolve separately from these additional tools. Although the discovery of one plugin by another plugin is mediated by the desktop, there is relatively little constraint on how these interactions are handled once established.

5.5.3.3 Composition

Declarative: The WCF composition of web services in the main system reduces the complexity of the overall system and provides for incremental recovery from failed client—server interactions. Increasingly, new features in the main system are being packages as "certified" plugins in order to simplify the extension of the main system and to simplify the system versioning problem. These certified plugins are functionally part of the main application, but they benefit from many of the simplifications resulting from loosely coupled SOA modules. Finally, the user can use the desktop plugin manager to select or deselect a plugin for inclusion in the user interface.

This selection process can be thought of as a (tool) composition process, which is controlled by the user. Some users may not need to use all of the functions in the system and this dynamic tool composition allows for a simplified user interface.

Procedural: As described already, the plugin machinery supports mechanisms for dynamic composition of algorithms and data. One commonly occurring example of procedural composition involves reader modules for field data, where there are competing vendor file formats, e.g. falling weight deflection data. The field data may be in any of a dozen data formats and each vendor will usually have the best reader for its format among competing readers. In this situation, we have been forced to allow plugins to be able to specify a precedence ordering over potential tool compositions to favor the vendor's reader over those of others. By supporting the composition of an ordered collection of readers, plugins can share a common set of reader code rather than needing to maintain separate versions of what should be substantially the same functionality.

Evolution: Again, the plugin machinery exists to support evolution of incremental parts of the system without a need to version the overall system. Also, the composition of algorithms and data from multiple sources (i.e. separate civil engineering firms) allows each plugin to leverage the shared inventory and reporting tools of the desktop.

5.5.3.4 Orchestration

Declarative: The desktop allows plugins to supply call back objects that implement required interface contracts for use when a user requests functionality from a plugin item. The desktop manages the user interface for the invocation of plugin functionality as well as loading an initializing the plugin. Also, plugins that plan to offer functionality to other plugins will usually define one or more interface contracts to facilitate the interoperation of the plugins. The desktop orchestrates the connection between the plugins and may not be involved going forward.

Procedural: Because these are decision support tools for practicing pavement engineers, a portion of the orchestration of the component services is actually done by the human user when implementing best practices in the domain. We may say that the human user helps to orchestrate the operation of the various services through the sequence of engineering tools used and through the various tool windows concurrently opened. The user interface supports the concurrent use of a collection of component tool windows, which may have originated from the main application or a plugin. The desktop also supports limited abilities to remember tool window configurations from one user session to another, which allows each user to control some of the orchestration settings across sessions, e.g. a user may request that a GIS map be opened on startup if available and the locations and sizes of tool windows can be remembered, which allows the user to play a role in the orchestration of tool windows.

Evolution: The evolution of the orchestration of the interaction of tools comes in part from the ability to add new tools with a variable number of user interface

elements for launching components of each tool or plugin. In addition, the ability of plugins to interoperate after they have been connected to each other by the desktop provides a path for the inclusion of new orchestration patterns by the inclusion of additional plugins. In fact, we are planning to mitigate the client versioning problem by the use of more and more certified plugins rather than main system versions. Because the orchestration of the underlying engineering algorithms depends, in part, on the user's selection of sequences of engineering operations and/or reporting parameters, there is room for the orchestration of the tools by the combination of the human user and the desktop to evolve with changing best practices.

5.6 Fractal Issues We Have Identified

Some of these issues were known to the authors prior to encountering them in the pavement management domain and some of them have emerged during the course of working in this domain. In this section, we will summarize the high level issues we found and relate them back to the real world example. We have identified the following overarching principles from our experience:

- Finiteness limits drive the need for increased structure—constraints on time and/or resources can require a more highly structured solution in order to solve the problem at hand within the given constraints;
- SOA Federations favor some structural patterns over others—some patterns of software structure produce better SOA Federations than others and there are some guiding principles for choice of good patterns;
- SOA Federations favor late binding—late binding to a particular solution element avoids the details of selecting the best component tool until it is actually needed;
- Mixed Initiative Dialog—a recognition that SOA systems, whether made up of human or machine actors interact more as peers and either side of the communication may have the initiative from time to time;
- Trust, Reliability, Ability, and Authoritative Source—when designing SOA systems or discovering a candidate for dynamic composition, potential human or software actors cannot always be treated equally.

For each of these issues, we provide a theoretical basis and a basis in our experience. These two items independently provide support for our conclusions about the fractal nature of these issues and their importance in SOA.

5.6.1 Finiteness Limits Drive the Need for Structure

Baskin et al. have explored this idea in detail in [32]. In the interest of keeping this material self-contained, we briefly summarize the key parts of this principle here.

Theoretical Basis: Think of a problem to solve in the form of a mathematical function, which is a subset of the domain of input values cross the range of output values. Any computable function can be represented by a universal Turing machine with a starting tape containing a definition of the finite state controller for the machine and a finite amount of starting data. The Kolmogorov complexity [33] of the function can be thought of as the length of the shortest universal Turing machine starting tape that solves the problem. Multiple minimal starting tape can exist with different structures. The size of the function can be taken to be the length of a starting tape with a simple finite state controller that exhaustively searches a table of input/output pairs and matches the given input value(s) to find the corresponding output value. If the size of the function is equal to the Kolmogorov complexity, then there is no room for the use of structure within the function to reduce the size of the starting tape and still solve the function. If the Kolmogorov complexity is less than the size of the problem, then there are some combinations of the various input values that lead to a common outcome. In this formulation, we can think of finiteness limits as limits on the run time of the universal Turing machine and/or limits on the length of the starting tape (i.e. limits on the size of the starting data and/or the state transition diagram for the finite state machine controller).

When finiteness limits are imposed, the nature of the solution must be modified from an exhaustive lookup table to exploit the commonality among subsets of the presenting inputs. This process innately involves the explicit incorporation of these patterns of commonality into the structure of the solution. The recognition of the commonality in the structure of the problem requires increased structure within the solution and that structure must exploit the patterns of commonality within the function. Finiteness limits may be expanded also. When finiteness limits are expanded, e.g. by increasing speed and/or expressive power of solution platforms, then either less highly structured solutions can be used or more complex functions can now be addressed within the expanded limits.

Domain Examples: The initial impetus for the integration of PAVER and PCASE into a single desktop came from the imposition of a finiteness limit, i.e. the push to replace two (partially redundant) inventory definitions with a single (unified) inventory structure. In a similar way, combining the functionality of these closely related tools matched the reduction in staffing (another finiteness limit), which accompanied the reduction in field engineering office head counts during the past twenty years. The more recent pursuit of enterprise level asset data management was stimulated by a DoD push to respond to pressure on budgets over the past decade and a need to be more efficient in the allocation of resources. The push for plugin modules in the Desktop represents an attempt to reduce the cost (a finiteness limit) of civil engineering tools by leveraging the common inventory and presentation tools so that each vendor's civil engineering tool does not need to have its own (redundantly expensive) version of these shared tools. In fact, it might be possible for some civil engineering tools to actually have a shorter development time, lower development cost, and increased range of functionality by leveraging

the desktop tools. In order to exploit these things, the complexity of the plugin tools must generally be increased by the need to conform to the structures demanded by the plugin framework, which is a layer of abstraction that could otherwise be avoided.

5.6.2 SOA Federations Favor Some Structural Patterns Over Others

As we saw in the previous section, some representations of the form of the solution can embody more knowledge about the structure of the problem at hand than others. Among competing structures for solutions, we have found that some structures are more useful than others. We have identified the following structural patterns, which are especially useful for constructing SOA federations, but are also good software engineering principles as well:

- Matching—the patterns of coupling and functional decomposition in the solution will be simplest and more robust if they match the analogous patterns in the problem domain itself;
- Favor Composition over Specialization—is a common adage in software engineering but it is especially useful in SOA federations because this bias makes discover and dynamic composition much easier;
- Manage Variation Explicitly—try to find a balance point between exhaustive enumeration of standards and the chaos that results from a lack of attention to the explicit management of patterns of variation;
- Manage patterns of coupling to maximize convergence of the SOA solution under change—the evolution of best practices means that large SOA systems will change and convergence under change is essential;

Theoretical Basis: The imposition of finiteness limits drives the increase in structure for solutions. We first encountered the notion of matching in biology [34] but we have actively employed it for decades now. Among competing structures for solutions, biology and, by analogy, best software practice, favors solutions whose structure matches the patterns of structure in the environment and in the structure of the problem. In software this means an analysis model where all object class names and relationships are recognizable by a domain expert as a model of the domain. The domain model will be more stable than any particular solution structure and will be better suited for evolution to solve related problems later. The principle of matching suggests that wherever there are components of the problem domain with differing rates of evolution, there should be a factor point (composition) to allow two components of the software to evolve separately. Similarly, when two components in the problem domain are highly coupled, then they should be coupled in the structure of the solution.

The virtue of composition as a tool for modular replaceable parts is well known. The notions of Discovery and Composition in SOA are intended to directly exploit composition. Composition is also a key tool for avoiding duplication of functionality in multiple places where the evolution of the functionality needs to be shared.

Explicitly managing variation is closely related to the principle of matching, which was described earlier. It involves finding boundaries in the domain where elements can vary separately and then match that boundary with a comparable software module boundary. Either enumerate the variety completely (e.g. Metric or English unit systems) or explicitly allow for variation in a constrained way, e.g. using in interface contract. Patterns in dynamical systems can also be shown to suggest that some structural patterns support evolution better than others [35].

First derived in mechanical engineering but also applied to software, the principles of Axiomatic Design [36] show that it is possible to form a dependency matrix describing patterns of coupling among software modules and describing the pattern of use of software modules to solve a problem. When the dependency matrix for the software modules can be made lower triangular, then there is a precedence ordering over the modules such that they are stably convergent under changes in them. Software modules and patterns of using them to solve a problem for which the coupling matrix cannot be made lower triangular require simultaneous and coupled changes in multiple places and are not convergent under change. Changes in software modules can be driven by changes in the requirements and changes in technology. Explicitly managing patterns of coupling facilitates software evolution.

Domain Examples: The object model for PAVER is a domain model, which is then used implement the various requirements for best practices. It matches the real world pavement domain and, hence, has evolved well. The origin of the enterprise systems as separate "smokestacks" also mirror the substantial separation of these functions. The origin PAVER as a separate system made it natural to see the engineering rules and analysis modules as an extension of the basic inventory. In hindsight, this use of specialization to add functions to the basic inventory was a mistake. We should have favored composition and, then, when PCASE came along, both PAVER and PCASE could have shared the inventory as a commonly held part. Unfortunately, it has not proven practical to fix this mistake and a workaround has been required. Had we favored composition originally even when there was not an obvious use for it, we would have had a better domain model and an easier time integrating PCASE with a shared inventory.

Managing variation explicitly has been done extensively in PAVER where certain things can be locked down via analytic closure (e.g. surfaces are flexible, rigid, or unpaved). In other places where the domain allows for meaningful variation and/or extension, then users are allowed to extend built-in types and are required to supply engineering attributes of these types so that they can be used by the analysis algorithms. The most interesting examples of explicitly managing variation come from the Enterprise Federation. The integration of GIS attributes between PAVER and the enterprise GIS attributes has been seen as a problem of locking down the attributes to be supplied by PAVER to the enterprise GIS system.

At first glance, a specifically enumerated set of attribute values would appear to make the problem simpler but it actually makes it harder. Especially in a GIS where users can define their own coloring strategies for attributes, there is constant demand for more and varied attributes from PAVER. The ultimate solution was to define a GUID in the GIS to be matched to a GUID in PAVER and then to have the GIS user "join" the GIS data to the PAVER data as needed. This explicit management of variation by identifying the only legitimate standardization and tolerating complete freedom after that point is simultaneously an instance of matching, explicit management of variation, and late binding as presented in the next section.

The integration of the real property/asset management data also provides an instance of matching, explicit management of variation, patterns of coupling and late binding. Both the pavement domain and the real property domain have an inventory hierarchy and the hierarchies are highly correlated. The original software requirements given to the PAVER development team were to modify PAVER to enforce the congruence of the pavement domain inventory hierarchy and the real property hierarchy. The high correlation of these two inventory structures (80 %) made this convergence appear to be simpler than allowing them to be different. By applying principles of domain modeling and matching, the PAVER development team was able to push back and get permission to implement the real property tags at a lower level than originally requested with the justification that there were legitimate domain rationales for an item to be at two incompatible places in the two hierarchies and by tagging lower level elements, it was possible to dynamically roll up the PAVER inventory according to tags representing a somewhat different composition. The system allowed tags to be supplied at the originally requested higher level and only exceptions to that assignment were needed at the lower level. This approach exactly matched the predicted pattern of domain variability. Despite repeated attempts to force field engineers to use the original exact correspondence and tag at the originally requested level, the system was not usable for some locations. Once users were allowed to tag exceptions at the lower level, the system was accepted. This is an example of predicting a domain requirement based upon fractal SOA principles and persevering in the face of resistance from contract monitors.

Explicit management of patterns of coupling is a combination of software development practices and domain modeling. It is possible to explicitly manage patterns of coupling by using a heavily constrained N-tier software development model. Each tier consists of an interface contracts module, which is visible only to the layer above, and an implementation module, which is not visible to the layer above. These constraints can be enforced by allowable patterns of reference among modules. Using this approach, it is not possible for implementation code in one layer to become coupled to implementation code in a lower layer. Using a strongly contract driven pattern of allowable coupling does lead to a larger number of modules than would otherwise be required. If needed, these decoupled modules can be merged at the last minute to ease packaging while retaining the constraints on coupling.

5.6.3 SOA Federations Favor Late Binding

As shown in the issues surrounding trust, the SOA ideas of discovery and composition imply a substantial departure from the historical notions of "link editing" all of the software modules in a software system into a single executable at load time. Like other issues we have identified, the notion of late binding applies at all levels of abstraction.

Theoretical Basis: The notion of late binding has actually been around for a long time. In the original LISP implementations, the boundary between program and data was blurred and a LISP program could build a list of instructions and then execute those instructions! We stop short of that ultimate example of late binding in our discussions here, but we note that the inclusion of human users as one of the SOA Federation members, we achieve late binding of a similar order because the human users may elect to compose data sources in entirely new ways and may interpretively execute algorithms by invocation of SOA Services where the algorithm exists only inside the user's mind or is codified in a best practices manual.

We find another model for late binding in what Lu et al. [37] has called Engineering as Collaborative Negotiation (ECN). The ECN paradigm was developed for mechanical engineering product design and we have applied some of its principles in our work. The basic idea of ECN is to identify constraints on the design result as early as possible but with the broadest tolerances possible. This approach is in contrast to more traditional mechanical engineering design approaches that emphasize the preparation of relatively specific designs for major subsystems as the overall product design matures. By identifying broad constraints on the design as early as possible and delaying selection of specific design values until as late as possible, designers can detect conflicts in the design much earlier than would otherwise be possible. This late binding approach leads to a more agile and cost effective design process. We find a similar situation in SOA Federations because the delay in binding to details can afford opportunities for opportunistic selection of tools for composition.

Domain Examples: The mechanisms for late binding are different at each level of abstraction, but there is value to late binding at each level. The matchup for GIS data and Real property data needs to be late bound because the Air Force, Navy, and Army all use PAVER but use different enterprise systems. We can use a single data harvest mechanism, but each service must use its own tools for accessing the data. Secondly, the data may be referenced in place (planned for the Air Force) or by accessing a "published" copy of data (Army and Navy).

The Desktop federation of PAVER and PCASE uses late binding when importing data and when exporting data. This federation has a collection of application tools, which may contain one tool or both tools depending upon what the user has installed. The import and export tools actually bind to the collection of applications for putting data into an export file or bringing in data from an import file. These operations are internalized into each of the Desktop SOA federation members. The data for all plugin modules is processed by the desktop for

import/export but each plugin module will only be bound to data if there is both data in the plugin persistence and the plugin is activated for the user. Another example of late binding occurs when the user opens a database. Any given database might have been created with PAVER alone, PCASE alone, or both PAVER and PCASE installed. During the file open process, each application is allowed the opportunity to create any missing databases, which might be exclusively managed by one tool and to add them to the single logically unified database. In this way, the number of databases and data tables is late bound at the time of file open.

Late binding is a fundamental part of the plugin machinery. Some plugin modules may be installed as an integral but optional module. We refer to these plugin modules as "certified" because they can be recognized as being compiled with the same trust level as the main application code. The user can opt to include these tools in the user interface and, thereby, cause a late binding of user visible functionality. For plugin modules from other civil engineering firms, there is both late binding of those tools to the desktop and also there can be late binding of these tools to each other through a SOA discovery and composition protocol, which is mediated by the Desktop as it orchestrates the initial setup of each plugin module.

5.6.4 SOA Federations Contain Mixed Initiative Dialogs

One reason that we have used the word federation throughout this discussion is the realization that we are bringing together a collection of relatively coequal participants where the union of what the participants know and can do is needed to solve complex problems. This relative symmetry of the participants gives rise to the situation where one federation member may predominate at one point in time and another participant at another time, and, hence, gives rise to the need to support a mixed initiative dialog at all levels of abstraction.

Theoretical Basis: We borrow the notion of a mixed initiative dialog from intelligent tutoring systems [38, 39] in which two semi-autonomous agents interact with each other while exchanging the control over the interaction. In intelligent tutoring systems, the human student is made to be more actively involved by being put into a position of active participation rather than passive listening. In our work on engineering field office automation in the 80's, we saw a clear role for supporting the field engineer with individual decision support systems and later groups of engineers with group decision support systems. By keeping the practicing engineer involved in the decision making process, some of the harder problems can be offloaded onto the human user and, thus, we arrive at expert support systems (where the expert is the human decision maker) rather than expert systems (where the computer system is expected to be an expert).

Domain Examples: In our civil engineering maintenance management tools, we emphasize supporting the decisions of a practicing pavement engineer rather than replacing their decision making with expert-derived rules or algorithms. Examples of this include the provision of common domain defaults for all required inputs for

work planning together with the ability for the engineer to override the defaults or extend them, e.g. an initial set of work types, surface types, cost tables, and budgets. The incorporation of plugin modules is another example of a mixed initiative dialog where in the installation of a plugin module requires administrative privileges but does not automatically activate the plugin for the user. By using the "add in manager" tool, the user can control which Add Ins (plugins) are actually presented in the user interface. Finally, the composition of functionality among different plugins requires a dialog between the desktop and the plugin to expose it to other modules, which, in turn, enter into another mixed initiative dialog to exchange data and/or provide composite calculations.

At the Enterprise SOA Federation level, we see a mixed initiative dialog between the systems in the federation where the authoritative source (Spatial Data Standard system or real property/asset system) might temporarily give way to an operational authoritative source like PAVER because a field engineer standing on the pavement may be a more trusted source for those data as a byproduct of the field data collection.

Within the SOA Federation containing PAVER and PCASE, we again see a mixed initiative dialog, which we are still trying to fully realize: one system indicates that the most effective intervention is reconstruction and the other designs the details of the new construction. Each member of this federation draws upon the shared inventory but has the initiative for capturing and processing largely disjoint sets of time series data and analyses.

While there is technically a mixed initiative dialog between the various web-services for PAVER itself, the late binding of add on modules is more interesting and subsumes these internal issues. The mixed initiative dialog begins with the user electing to use the "Add In Manager" tool to activate one or more installed additional modules. Each plugin module has the option at this time to require a license key from the user and either it agrees to become activated or not. Once the plugin module has been activated, it can request that the Desktop host user interface items by which the user can request the plugin to respond. The Desktop and the human user have the initiative more often going forward, but a plugin module can do things like monitor a GPS feed and/or host its own user interface and the result of these data can be pushed back into the Desktop Federation to shift the user's focus of attention to see the newly selected data. This bi-directional control is most visible when there are two GIS maps being synchronized bi-directionally—one from PAVER and one from a plugin. To the human user, the two maps are part of a single unified interaction experience but the communication among the SOA Tool Federation members is not a seamless as it appears to the user.

Finally, plugin modules can call upon other plugin modules in something approximating the normal SOA Discovery and Composition techniques. During the instantiation of the plugin module, the Desktop and the plugin engage in a back and forth dialog by which plugin modules can agree to allow them to be used by others and a link between the modules can be directly established. Plugin modules that communicate directly with each other no longer need the Desktop to mediate further communication. Because these modules may also have their own user

interaction items, they behave more like co-routines than procedure call services and, again, we find a mixed initiative dialog between the plugin modules.

5.6.5 SOA Federations Depend upon the Explicit Management of Trust, Reliability, and Authoritative Source

As the length of this section title suggests, management of the issues surrounding trust is a somewhat messy problem. It involves issues between humans only, between software modules only, and between software systems and human users of those systems. As we will see below, this is also an issue between software development teams whose software systems will be members of a SOA Federation.

Theoretical Basis: The distinction between Authentication (do I believe you are who you claim to be) and Authorization (what I allow that authenticated identity to do) is well established, and we build upon that as a foundation for related issues. Implicit in the SOA notions of Discovery and Composition is the ability to select among competing sources for providing a required service based upon things like performance and competence. Rephrasing this in terms of trust, we get the question: Can I trust you to provide correct services/data in a timely fashion? A subtly related point can be used to limit which features are shown to human users: Can I trust you to be able to understand this feature and not be overwhelmed by too many features? Both of these two questions can be thought of as complementary aspects of the notion of competence—competence to provide and to consume.

Reliability closely relates to competence: (1) Can I rely upon you to provide good data/services in a timely fashion? (2) Can I rely upon you to understand and not corrupt the data and services I expose to you? These questions apply equally between all SOA Federation members whether they be a human or a computer software modules.

During the work on the enterprise federation of civil engineering tools for DoD, we encountered the notion of "authoritative source," which means the agency and/or software system designated as the official "go to" source for a body or data and/or expertise. We have coined the notion of "operational authoritative source," which means a source different from the officially designated source but, at least temporarily, better able to supply reliable data at a particular place and time. As we will see in the domain examples, these two competing sources will be separates SOA Federation members and they will enter into a mixed initiative dialog whereby the authoritative source may be updated by the transient activity of the operational source.

Although we are still trying to fully understand the fractal nature of trust issues, we have identified a key role for trust in the following substantially separable areas:

- Trust between software development teams for different SOA Federation members (federates), which involves territoriality, use of tools you cannot control, schedule compatibility, and competence;
- Trust between federates and their sources, which involves authoritative sources, perceived competence/timeliness of the services, and dependence on services whose availability cannot be guaranteed;
- Trust in the longevity of the available SOA services, which is critical in civil engineering data management where data must be kept over decades and the nominal life of the asset may be 50–100 years;
- Trust in user competence, which causes tension between those managing low level data and managers who must necessarily look across data from multiple sites/facilities and may not understand low level data.

These diverse aspects of the concept of trust can be found in surprisingly diverse situations and frequently prove to be deceptively simple to identify and incredibly difficult to resolve.

Domain Examples: The designation of an authoritative source is entirely external to the issues being discussed here but the existing authoritative sources for GIS data and Real Property/Asset data constitute a constraint on major aspects of the Enterprise SOA Federation. Although these systems are the authoritative source, PAVER can frequently be asked to provide updated data of higher quality as a byproduct of field surveys. In fact, many field surveys using PAVER are now required to obtain the latest GIS and Real Property data and for use in the inspection process and to return for potential update of the authoritative sources. This mixed initiative dialog between the systems may be fully or partially automated, but the authoritative source is responsible for the eventual integrity of the data and may refine or refuse proposed updates. At the enterprise level, trust between software development groups and SOA Federation members tends to be resolved by designation of by a single shared authority, i.e. the Secretary of Defense. The existence of a single "owner" for all of the SOA Federation members is on a panacea for resolving trust issues but it is a surprisingly underappreciated necessary condition. At the enterprise level, the issue of trust for the competence of the various SOA Federation members has been resolved by their separate evolution and individual validation. Again, the existence of a common governing authority with control over allocation of resources and the independently justified existence of the systems means that the longevity problem is also solved.

The development of the original Federation of PAVER and PCASE, which predated the inclusion of SOA principles, involved a process of reconciling the respective pavement engineering domain ontologies and establishing trust between the separate development groups (one made up of government employees for PCASE and the PAVER development team). Because there is a natural process of turnover in software teams, and because these civil engineering systems deal with problems over a period of decades, it has proven surprisingly difficult to maintain this trust between the groups over time.

The trust issues between plugins provide the richest examples of trust issues in the three layers of abstraction. When plugin modules can originate with the PAVER development team or with competing civil engineering firms there can be no illusion of a common controlling authority and no illusion that these SOA Federation members will always be there because plugin modules may be separately developed, licensed and distributed. Increasingly, these plugin modules depend upon external web services for data and operations and, again, continuity of access is qualitatively lower than for modules distributed as part of the main system.

The issue of trust is not an absolute distinction. An external authority can designate an authoritative source, but that distinction is artificial and external to all of the issues of the SOA Federation. The trust issues generally exist on a precedence ordering and not as a crisp distinction. We can illustrate this point with the following detailed example.

5.7 Conclusions

This chapter summarizes insights gained from more than twenty years of software development, maintenance, and evolution of a major pavement management system (PAVER™). We consider the traditional SOA concepts: (1) Ontologies, (2) Discovery, (3) Composition, and (4) Orchestration. Often, discussions of SOA techniques focus on stateless operations, which certainly have their place. Managing the persistence of time series data is essential whether orchestrating the collaboration of human civil engineers in a technology mediated federation or managing diet/exercise data with an app on a cell phone. Accordingly, we have cited this as a cross-cutting concern. We conclude that time series trends in such data are much more meaningful than any single snapshot of data. This observation leads us to an additional dimension, which cuts across all of the SOA concepts above: Algorithms versus Persistent State. Finally, during twenty years of experience in the pavement management domain, we have become attuned to the issue of evolution of best practices and associated decision support software systems, as well as the need for SOA Federations to support this evolution. This fact gives rise to a third dimension which we have explored in this work: Evolution. After reviewing our experiences with SOA Federations at three levels of abstraction, we have found the following basic principles to be self-similar at three levels of abstraction:

- Finiteness limits on time, participants, and/or resources demand more highly structured solutions;
- SOA Federations work best when they (a) match patterns of coupling/evolution in the domain, (b) favor composition, (c) manage variation as a first class issue, and (d) explicitly manage patterns of coupling;
- SOA Federations benefit greatly from late binding—especially when paired with management of variation;

- SOA Federations work best when there is a mixed initiative dialog among federates and human users;
- Trust issues must be managed by SOA Federations and among software development teams.

We have found all of these fractal principles in our historical review and we have used them prospectively to guide the development of new SOA system federates to good effect.

References

1. Shahin, M.Y.: Pavement Management for Airports, Roads, and Parking Lots. Chapman & Hall, New York (1994)
2. Reinke, R., et al.: Domain frameworks for collaborative systems: lessons learned from engineering maintenance management. CTS **2007**, 396–405 (2007). doi:10.1109/CTS.2007.4621780
3. Zdun, U.: Pattern-based design of a service-oriented middleware for remote object federations. ACM Trans. Intern. Tech. **8**, 3, Article 15 (2008). doi:10.1145/1361186.1361191
4. Li, Z., Cai, W., Turner, S.J., Pan, K.: Federate migration in a service oriented HLA RTI. 11th IEEE Symposium on Distributed Simulation and Real-Time Application, pp. 113–121. doi:10.1109/DS-RT.2007.31
5. Wang, W., Yu, W., Li, Q, Wang, W., Liu, X.: Service-oriented high level architecture. In: Proceedings of summer computer simulation conference, 2008. Article 16
6. IEEE: Standard 1516 (HLA Rules), 1516.1 (Federate Interface Specification) and 1516.2 (Object Model Template), September 2000
7. WSDL: Web services description language (WSDL) Version 2.0 Part 1: Core Language http://www.w3.org/TR/wsdl20/. Accessed 20 Mar 2014
8. SOAP: SOAP Version 1.2 Part 0: Primer (Second Edition) http://www.w3.org/TR/2007/REC-soap12-part0-20070427/
9. XML Schema: XML Schema Part 1: Structures Second Edition http://www.w3.org/TR/xmlschema-1/. Accessed 20 June 32013
10. Seo, C., Zeigler, B.P.: Simulation model standardization through web services: interoperation and federation on the DEVS/SOA platform. In: Proceedings of symposium on theory of modeling and simulation—DEVS integrative M&S symposium, 2012. Article 46
11. Li, J., Karp, A.H.: Access control for the services oriented architecture. In: Proceedings of ACM workshop on secure web services, pp. 9–17 (2007). doi:10.1145/1214418.1314421
12. specs@openid.net. "OpenID Authentication 2.0 Final." 2007. Available online at http://openid.net/developers/specs/
13. Liberty Alliance Project: Liberty ID-WSF web services framework overview. Version 1.1, 2005. Available online at http://www.projectliberty.org/liberty/specifications__1
14. OASIS: Web services security: WS-security core specification 1.1. OASIS Standard, 2006. Available online at http://docs.oasis-open.org/wss/v1.1/
15. OASIS: Security assertion markup language (SAML) 2.0 Technical Overview, Working Draft 05', 10 May 2005. http://www.oasisopen.org/committees/download.php/12549/sstc-saml-techoverview-2%5B1%5D.0-draft-05.pdf
16. Thomas, I., Meinel, C.: An identity provider to manage reliable digital identities for SOA and the web. In: Proceedings of IDtrust '10, pp. 26–36 (2010). doi:10.1145/1750389.1750393
17. Hatameyama, M.: Federation proxy for cross domain identity federation. In: Proceedings of DIM '09, 13 November 2009, pp. 53–62. doi:10.1145/1655028.1655041

18. Anastasi, G.F., Carlini, E., Dazzi, P.: Smart cloud federation simulations with CloudSim. In: Proceedings of ORMACloud'13, June 17, 2013, pp. 9–16 (2013). doi:10.1145/2465823.2465828
19. Al-Masri, E., Mahmoud, Q.H.: Identifying client goals for web service discovery. 2013 IEEE international conference on services computing 2009, pp. 202–209. doi:10.1109/SCC.2009.60
20. Dabrowski, M., Pacyna, P.: Cross-identifier domain discovery service for unrelated user identities. In: Proceedings of the 4th ACM workshop on digital identity management, pp. 81–88 (2008). doi:10.1145/1456424.1456438
21. Tolk, A., Turnitsa, C.D., Diallo, S.Y.: Model-based alignment and orchestration of heterogeneous homeland security applications enabling composition of system of systems. In: Henderson, S.G., Biller, B., Hsieh, M-H., Shortle, J., Tew, J.D., Barton, R.R. (eds.) IEEE winter simulation conference, Dec 2007, pp. 842–850. doi:10.1109/WSC.2007.4419680
22. Tolk, A., Diallo, S.Y., Turnitsa, C.D.: Mathematical models towards self-organizing formal federation languages based on conceptual models of information exchange capabilities. In: Mason, S.J., Hill, R.R., Mönch, L., Rose, O., Jefferson, T., Fowler, J.W. (eds.) IEEE winter simulation conference, Dec 2008, pp. 966–974. doi:10.1109/WSC.2008.4736163
23. Rathnam, T., Paredis, C.J.J.: Developing federation object models using ontologies. In: Ingalls, R.G., Rossetti, M.D., Smith, J.S., Peters, B.A. (eds.) Proceedings of the IEEE 2004 Winter Simulation Conference, pp. 1054–1062 (2004). doi:10.1109/WSC.2004.1371429
24. Calvanese, D., De Giacomo, G., Montali, M.: Foundations of data-aware process analysis: a database theory perspective. In: Proceedings of PODS '13, 22–27 June 2013. doi:10.1145/2463664.2467796
25. Reichert, M.: Process and data: two sides of the same coin? In Proceedings of the On the Move Confederated International Conference (OTM 2012), volume 7565 of Lecture Notes in Computer Science, 2–19 (2012)
26. Dobos, L., Csabai, I., Szalay, A.S., Budavári, T., Li, N.: Graywulf: a platform for federated scientific data and services. Proceedings of SSDBM '13, July 29–31 2013, Baltimore, MD, USA, 2013 ACM 978-1-4503-1921-8/13/07 (Pázmány Péter sétány)
27. Krizevnik, M., Juric, M.B.: Improved SOA persistence architectural model. ACM SIGSOFT Newsletter **35**(3), 1–8 (2010). doi:10.1145/1764810.1764821
28. Williams, K., Daniel, B.: An introduction to service data objects. Java Developer's J. (2004)
29. Carey, M.: The BEA AquaLogic Data Services Platform. Proceedings of SIGMOD 2006, June 27–29, 2006, Chicago, Illinois, USA. Copyright 2006 ACM 1-59593-256
30. Takatsuka, H., et al.: Design and implementation of rule-based framework for context-aware services with web services. In: Proceedings of iiWAS '14, 4–6 December 2014, Hanoi, Vietnam. doi:10.1145/2684200.2684310
31. Sarelo, K.: A SOA for ubiquitous communication management. In: Proceedings of iiWAS2009, 14–16 December 2009, Kuala Lumpur, Malaysia. doi:10.1145/1806338.1806386
32. Baskin, A., et al.: Exploring the role of finiteness in the emergence of structure. In: Mittenthal, J., Baskin, A. (eds.) The principles of organization in organisms. Santa Fe Institute studies in the sciences of complexity, Proceedings vol 13. Addison-Wesley, Reading, pp. 337–377 (1992)
33. Li, M., Vitanyi, P.M.B.: Two decades of applied Kolmogorov complexity: in memoriam of Andrei Nikolaevich Kolmogorov 1903–1987. In: Proceedings of 3rd annual structure in complexity theory conference, Georgetown University, Washington, 14–17 June 1988
34. Mittenthal, J.E., et al.: Patterns of structure and their evolution in the organization of organisms: modules, matching, and compaction. In: Mittenthal, J., Baskin, A. (eds.) The principles of organization in organisms. Santa Fe Institute studies in the sciences of complexity, Proceedings vol. 13. Addison-Wesley, Reading, pp 321–332 (1992)
35. Kauffman, S.A.: The sciences of complexity and "origins of order". In: Mittenthal, J., Baskin, A. (eds.) The principles of organization in organisms. Santa Fe Institute studies in the sciences of complexity, Proceedings vol 13. Addison-Wesley, Reading, pp. 303–319 (1992)
36. Suh, N.P.: Axiomatic Design. Oxford University Press, New York (2001)

37. Lu, S.C.Y., et al.: A scientific foundation of collaborative engineering. CIRP Ann. Manufact. Technol. **56**(2), 605–634 (2007). doi:10.1016/j.cirp.2007.10.010
38. Chan, T-W., Baskin, A.: Studying with the prince: the computer as a learning companion. In Proceedings of the ITS-88 Conference (1988), pp. 194–200
39. Graesser, A.C., et al.: AutoTutor: an intelligent tutoring system with mixed-initiative dialogue. IEEE Trans. Educ. **48**(4), 612–618 (2005). doi:10.1109/TE.2005.856149