SWiFT: A Tool for Constructing Workflows for Dynamic Network Analysis

David Garlan, Bradley Schmerl, Vishal Dwivedi, Aparup Banerjee, Laura Glendenning, Mai Nakayama, Nina Patel
School of Computer Science, Carnegie Mellon University
Pittsburgh, PA, USA, 15213
{garlan,schmerl,vdwivedi}@cs.cmu.edu, {aparup,lglendenning,m2.nakayam,nina.patel.82}@gmail.com

***Abstract*—Dynamic Network Analysis is a domain of computation that refers to the analysis of complex social systems that change over time. Within this domain, analysts need to be able to carry out workflows, involving composition of various tools and procedures that they use to extract and analyze social systems. Typically these workflows require the incorporation of heterogeneous tools in distributed locations, are interactive in that they allow dynamic parameterization and exploration, and can be shared with and reused by other analysts in different contexts. Furthermore, because such analysts are not computer programmers, they require an approach for constructing workflows that allows them to focus on their analysis task, and that provides them with appropriate guidance in constructing and reusing workflows. In this paper, we describe a tool, called SWiFT, that provides a workflow construction environment for dynamic network analysis, built on a service oriented architecture, that provides these features.**

Keywords-dynamic network analysis, workflows, service oriented architectures.

I. INTRODUCTION

An important emerging computationally-centered domain is the study of complex information using Dynamic Network Analysis (DNA). DNA refers to the analysis of rich relational models that represent entities, relations, their properties, and how all of those change over time. For example, social networks are a special case involving people and relationships. Other more complex networks may include entities like locations, ideas, times, artifacts, etc. and corresponding relations [1][2].

DNA is increasingly being used in a variety of fields including anthropology, sociology, business planning, law enforcement, and national security. Analysts in these fields typically extract entities and relations from unstructured text (such as web sites, blogs, email, etc.) to create network models, and then use those models to gain insight about social phenomena through analysis and simulation.

This work was supported in part by the Office of Naval Research - ONR-N000140811223, and the center for Computational Analysis of Social and Organizational Systems (CASOS). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, or the U.S. government.

Over the past decade a number of powerful tools have been developed to assist DNA analysts, and many more are being built every day. Such tools support extraction of models from a variety of unstructured text formats; provide mechanisms to massage, filter, analyze, and visualize networks and the information extracted from them; and enable simulation of behavior and investigation of "what if" scenarios.

Unfortunately, today these tools tend to run as stand-alone interactive desktop applications. This has a number of negative consequences. Each tool typically has an idiosyncratic interface with a steep learning curve. Composition of functionality from several tools is often not possible. And when it can be accomplished it is usually done in an ad hoc fashion using intermediary files and scripts, leading to brittle configurations and requiring analysts to have detailed, low-level knowledge of model formats and storage conventions. Analyses, once performed, are hard to share with other analysts, or to even record in a repeatable fashion what was done to obtain some analytical result from a given input. Evidentially what is needed is a way for non-technical analysis (i.e., those without detailed knowledge of tool implementation, file systems, or programming) to combine capabilities of existing and future analysis tools in a flexible, intuitive, and repeatable way.

Similar problems have been addressed in other domains such as scientific computing. However, as we detail later, DNA has a number of requirements that in combination make such existing solutions inappropriate. These include the need to support (a) end-user composition of a wide variety of tools running on many different platforms in a distributed setting; (b) a variety of analysis interaction modes – from "pushbutton" operation to highly interactive exploration; (c) task-oriented guidance to help analysts create appropriate analysis compositions; and (d) reusable analysis templates that can be shared with others and flexibly tailored to a new analysis tasks.

In this paper we describe a system, called SWiFT, that addresses these requirements.

SWiFT provides a web-based graphical workflow construction environment, similar in spirit to mash-up languages like Yahoo! Pipes, but houses this on top of a SOA that supports composition of heterogeneous tools running in a distributed setting. Key to this environment are the engineering decisions that allow SWiFT to support a variety of interaction modes (including use of desktop applications) within a uniform workflow model; provide extensible, task-oriented assistance

(such as mismatch repair, privacy analysis, and machine-learning based guidance); and provide the ability to create sharable analysis templates that can be tailored to new contexts.

II. Dynamic Network Analysis

A. Context

Dynamic network analysis involves understanding, analyzing and predicting relationships in complex social systems. Dynamic multi-mode social networks are typically used to represent these social systems. These networks relate entities in the system (e.g., people, knowledge, places, actions) to each other. Within this field, computational techniques, such as machine learning and artificial intelligence, are combined with traditional graph social network theory, and empirical research on human individual and group behavior to develop and test tools, theories and actions that are enabled and constrained by relations in the network.

Dynamic network analysis typically entails a series of steps. First, an analyst gathers data from disparate sources. One common approach to this is to extract networks from a corpus of texts, such as web pages, news articles, journal papers, stock holder reports, community rosters, etc. Second, the extracted networks need to be analyzed. That is, given the data, analysis identifies key actors and sub-groups, points of vulnerability, and so on. Third, given a set of vulnerabilities, an analyst may ask what would happen to the networks were the vulnerabilities to be exploited. How might the networks change with and without strategic intervention? This kind of analysis is similar across different sub-communities of dynamic network analysis. For example, military intelligence may use news reports, intelligence reports, etc., to build a network to understand the "human terrain" of a field of operations, and then use simulation to determine how best to communicate with a population; federal agencies may use news stories and crime reports to understand gang-related drug activities in a city and use simulation to plan the best courses of action to fight them; sociologists may use interviews and other sources to understand a population and then use simulation to work out the best strategies for educating them about policy changes.

Unfortunatley, today it is difficult to capture common sets of procedures so that they can be re-executed, shared, and reused by analysts in different settings. Analysts typically need to be expert users of a range of existing tools, and need to remember what they did in each of these tools in order to reproduce results, or train new analysts. Thus, an approach that allows analysts to specify these common *workflows*, reproduce them, or share them with others who might use them in different contexts is sorely needed.

B. Domain Requirements

The domain of dynamic network analysis, has a number of requirements, the combination of which pose a set of challenging engineering problems for automated support.

1) Compositionality

A capability allowing analysts to automate their tasks is one that enables different analysis steps to be composed into larger procedures. Within dynamic network analysis, a large variety of tools have been developed to help analysts produce and analyze networks. For example, the Analyst's Notebook (I2)[2], ORA [3], and UCINET [4] are tools to help conduct network analysis. Additionally analysts use general-purpose tools for activities like web scraping, text mining, data mining, statistical analysis, geo-spatial analysis, decision support, simulation, and gaming. However, these tools are typically coarse-grained and feature-rich (often running as desktop applications), and they do not provide support for automating steps of an analyst's task that require linking a set of such tools. This leads to the compositionality requirements of:

a. *Heterogeneity.* Having invested time and money in existing tools, analysts are reluctant to abandon those tools. Rather, they desire to easily compose the functionality of different existing tools. Some of the tools are highly interactive; some tools provide simple batch procedures for processing text or annotating networks with more information. Analysts need to use a wide variety of tools written by multiple organizations running on a variety of platforms. Furthermore, we require a platform on which the fine-grained *steps* of an analyst's workflow can be executed individually. We therefore need an approach that allows us to decompose the tools into their constituent features. Furthermore, analysts want to be able to use the functions of diverse set of tools in conjunction to provide end-to-end analysis. Composition, therefore, needs to involve a mixture of batch-oriented procedures and interactive standalone tools.

b. *Orchestration.* In addition to having access to a wide variety of tools, analysts must be able to compose them as workflows that can be automatically executed. Ideally these workflows would match the pattern of steps that an analyst might manually execute, directly reflecting the high-level flow of analysis.

2) Interactivity

The range of skills of analysts is quite diverse, and so automation needs to cater to different levels of interaction. Along this spectrum, we identified three key interactivity requirements:

a. *Turnkey automation.* Some novice analysts simply want to use existing workflows produced by more expert analysts to produce reports that are consumed by their superiors. These can be considered as batch processes, where the analyst can specify input data, and simply produce results.

b. *Parameterization.* In some cases it is necessary to ask the analyst to enter parameters dynamically as a workflow executes. For example, in a workflow that generates a network and produces a report, the user may be asked to specify which agents to focus on in the report. The set of agents to choose from will only be known once the network is generated, and so it is not possible to enter this information in the manner described in (a). It is therefore necessary that a workflow be able to ask for and receive this information from the user as part of its execution.

c. *Exploration.* Another sub-community includes analysts wo are authoring novel workflows. In such instances, analysts

[2] www.i2group.com/us/products-services/analysis-product-line/analysts-notebook

are exploring the design space for workflows, setting up correct data for processing, and testing the workflows. In such a mode, users may want to pause the workflow and inspect intermediate data produced as part of the workflow to ensure that steps have executed correctly. In fact, some analysts may want to examine intermediate results to determine the source of conclusions made by steps later in the workflow, even as part of their normal workflow execution. Furthermore, analysts will want to interact with the interfaces of their existing tools to conduct additional analysis, finesse presentations and reports, and gain further insight into the results.

3) Guidance

In order to further support analysts, human-centered guidance is required to and provide advice on solving these problems. This requirement of guidance can be divided into two areas:

a. *Authoring guidance*, which advises analysts in creating their workflows. Such guidance might range from simple syntactic checks, to advice on common ordering of steps, to providing advice on how to repair a workflow in the presence of data mismatches.

b. *Execution guidance,* which advises analysts on choosing appropriate workflows for their tasks. The meaning of "appropriate" may differ in different contexts, but it ranges from providing information about common workflows for given sets of data, to execution quality of service advice for qualities such as performance, security, provenance, etc.

4) Sharing and Reuse

As noted earlier, a key requirement is to allow analysts to specify workflows that can be shared with other analysts and reused in similar contexts. For example, an analyst might define a workflow to analyze the disaster relief effort immediately following the 2010 earthquake in Haiti. When a nearly identical problem arises, such as analyzing humanitarian relief for the Chile earthquake also in 2010, the (different) analyst should be able to locate this workflow and use it in this new context, but with slightly different data. For example, the source data for Chile may have been scraped from the web instead of from a news source as for Haiti, and requiring additional steps to remove HTML tags.

While today's analysts can share the results of different analyses (for example, through standardized reports), sharing the workflows that enabled those analyses is almost impossible because typically there is no formal description of those workflows or mechanisms to repeat them verbatim, tailor them to a new context, or incorporate them into other workflows.

In summary, dynamic network analysts require tool support for composing workflows to automate their analysis tasks. Such tasks involve a heterogeneous mixture of fine-grained steps and feature rich tools. Workflows need to support a number of interactivity modes. Tool support needs to enable sharing and reusing these workflows with other analysts, and in different contexts. Finally, domain-specific guidance in constructing and executing workflows is required.

III. Related Work

The use of workflow compositions is not new to the DNA domain. However, what is different is the requirement for heterogeneity and the interactivity, and the various guidance mechanisms, and capabilities for sharing and reuse of workflows. In this section we trace how workflow composition tools used in other domains have addressed such concerns.

A. Service Oriented Architecture (SOA)

Service-Oriented Architecture (SOA) is a widely used mechanism for developing large-scale distributed systems through the composition of loosely-coupled services available over the Internet. To migrate various heterogeneous systems into the SOA domain, they are typically modified (often through wrappers or adapters) to provide a service-based interface, which is registered with a SOA. These services can then be invoked using standard web protocols such as SOAP or REST, or they can be combined with other services to produce more complex applications. SOA enables such compositions via languages such as BPEL, BPML and XLANG that can be executed in an orchestration [5].

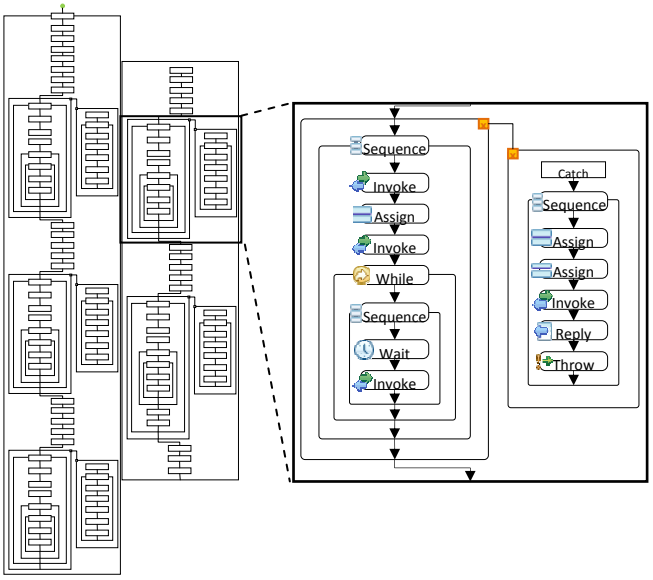

Figure 1. BPEL orchestrations for some standard DNA tasks

However, these languages and the other SOA infrastructure services (such as registries and support for provenance) require writing scripts that are typically too low-level for human-centered computation where analysts who write them are technically naïve. For example, as illustrated in Figure 1, even a relatively simple workflow (here containing five logical analysis steps) requires the specification of many BPEL steps involving assignment of variables to prepare a service invocation, invoking the service using the service APIs, and dealing with errors. Not only is writing such representations difficult for non-technical users, but they also impose additional complications such as inability in dealing with user interaction and lack of guidance beyond syntactic checks.

Current SOA based composition languages were designed for automated execution in business scenarios, and not for de-

sign by end-users. It is therefore not surprising that they lack capabilities to incorporate human interactions during the execution of workflows. BPEL4People [6] is a recent proposal to address the problem of interaction, by providing a special- purpose construct named WS-Human-Task, but it still does not address the overall issue of the complexity of a BPEL process.

As we discuss later, in our implementation of SWIFT - we build on SOA, but add additional layers to bridge the gap between technology and users.

B. Scientific Computing

Scientific Computing, in contrast to SOA provides high-level domain-specific support for composition – primarily through workflows. These workflows involve executing pipelines of parallel data-processing implemented by heterogeneous tools, as opposed to procedural control flow executions in BPEL [7]. Scientific workflows on the other hand, contain many tasks, involve large data sets, and often require support for analysis and provenance that requires iterative execution of workflows rather than an automated turn-key based execution as in BPEL.

It is also quite common in scientific computing applications to have workflows involving heterogeneous, independent components, running in a distributed setting. Examples of such systems include MeDICi [8] and VistA [9] in the healthcare domain, SORASCS [10] in the social-sciences domain, and Galaxy [11] for genome and bio-informatics.

Recently in the scientific community there has been some effort to integrate data and computation resources using SOA infrastructure. For example, Taverna [12] and Kepler [13] are popular workflow design that can integrate web-services developed for scientific computing. This has led to the existence of eco-systems that promote sharing of workflows and web services through communities such as MyExperiment [14] or through service registries such as Biocatalogue [15] where users can register web-services for others' use.

However, most workflow tools in the scientific computing domain offer only limited support for interactivity. Also, the existing scientific-computing platforms assume homogenous granularity of components that are composed– these are either web services, or tool APIs, but never both, as scenarios that involve mix and match of capabilities of tools and services are simply not supported.

C. Mashups

Mashups are applications that combine data, presentation or functionality from two or more sources to create new services. *Data-mashups* in particular, can combine similar types of media and information from multiple sources into a single representation. Mashups support composition and distribution, but lack support for combining heterogeneous data-sources, as their model is mostly based on pull-based mechanisms relying on RSS or Atom feeds.

Unlike workflow composition tools, mashups have been successful in handling interactivity by providing support for parameterization and displaying intermediate results. One popular mashup tool is Yahoo! Pipes [16] that inspired the design of SWIFT.

D. Dynamic Network Analysis (DNA)

As we discussed before, most DNA tools today involve executing analysis procedures implemented by various standalone tools. Currently it is extremely difficult to compose functionalities across different tools as they often involve human-centered workflows using intermediary steps of data-processing, and sharing. Some tools provide scripting interfaces to automate some parts of the dynamic network analysis task. For example, Automap [17] provides a scripting interface for processing raw text into networks, organizing steps into phases such as ingestion, cleaning, processing, generation and so on. However, such support is limited to only the functionality provided by a single tool. Moreover, such scripting does not support the requirements for interactivity, outlined earlier. Nor does it support the requirements of guidance.

IV. SWiFT

SWiFT is a tool to support DNA, that is built on top of the Service ORiented Architecture for Socio-Cultural Systems (SORASCS) [10]. SORASCS incorporates standard open-source SOA technologies (mostly from the Apache Foundation). SORASCS provides access to heterogeneous services for dynamic network analysis, that may be either fine-grained processing steps that run on remote servers, or as standalone tools that run on a client's machine. As such, it addresses the requirements of heterogeneity, outline above.

However, as we argued earlier, SOA technology by itself is not sufficient. To bridge the gap and thereby directly support human-centered computing in the DNA domain, SWiFT provides a user interface, inspired by Yahoo! Pipes, that allows SORASCS services to be connected together into workflows. These workflows are then compiled into BPEL orchestrations, which can then be registered as new services in SORASCS. Once registered, they can be shared and reused just like any other services provided by SORASCS. In this section, we describe the user interface features of SWiFT, and in Section V we elaborate on the underlying architecture and implementation challenges in realizing this interface.

The user interface of SWiFT is organized around a design based on *perspectives*. Currently there are three perspectives – Compose, Execute, and Analyze – which organize the different UI features into related categories. Figure 2 shows a screenshot of SWiFT in the compose perspective. Users can easily switch between the perspectives using a tabbed widget, and each perspective uses a similar layout consisting of a central *canvas*, showing the workflow(s) currently being worked on, toolbars, and other panels as needed. The following sections discuss the perspectives and how they address the requirements and challenges previously discussed.

A. Compose Perspective

The compose perspective allows a user to compose and orchestrate a workflow from services. Services that are available to the user on SORASCS are displayed in a *palette*; these services can then be dragged-and-dropped onto the canvas. The user can use a number of heterogeneous tools as components in their workflow, such as primitive services, saved workflows, and composite services. On the canvas, these components can then be connected and ordered to form a fully orchestrated

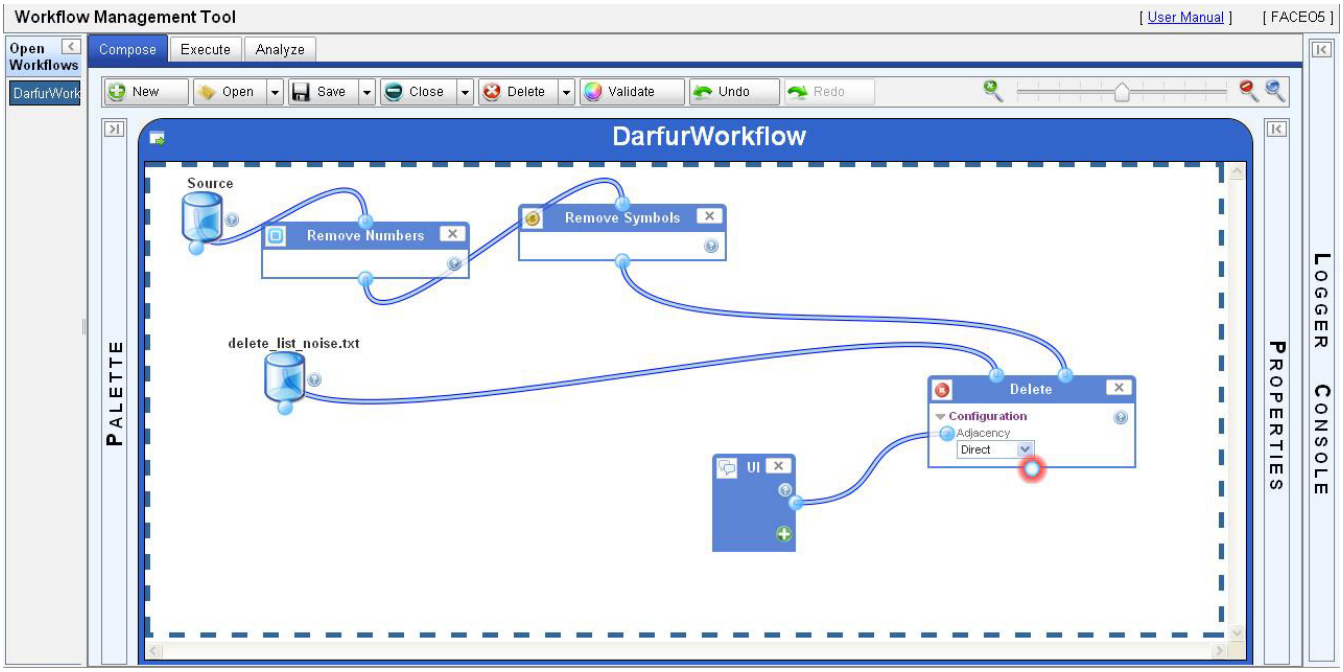

Figure 1. SWiFT Compose Perspective.

workflow. Figure 2 shows an orchestration containing three primitive services, two pieces of data, and a special service called the UI service (described later).

From this perspective the user can create new workflows, edit existing workflows, and save their workflows. They have access to not only their own work, but also any shared workflows for which they have permissions in SORASCS. The goal is to promote building upon the expertise of others by having knowledge repositories where users can share workflows and data with others.

To facilitate the reuse and sharing of workflows, the compose perspective also includes features for *packaging* a workflow as a reusable entity. This involves three steps: 1) *expose* internal ports and parameters of services inside the workflow to the workflow's boundary, 2) give the workflow meta-information such as a name, description, and tags, and 3) set the permissions of the workflow to specify who will have read-access and write-access to it.

The SWiFT palette orders services hierarchically based on meta-data such as tags and service type. This categorization hierarchy can be changed by the user to dynamically reorganize the services. The palette also provides searching and sorting capabilities. Because the number of services is expected to grow large, it is important to have UI features to allow users to easily find and user the particular service(s) that he or she wants. Once found, another useful feature is to mark a service as a *favorite*, which then shows up in the favorites section of the palette for easy access.

The compose perspective also has some guidance in terms of syntactic checks of the workflow. Before a workflow can be executed, it must be syntactically valid. SWiFT has a number of rules that define what a "valid" workflow is, and these are interactively presented to the user in terms of syntactic checks. Some of these checks happen while the user is composing a workflow; for example, if a user tries to connect two ports that have different types, the port will glow red and the user cannot connect them. Some of the checks happen when the user validates the workflow or tries to execute it; for example, a workflow is not valid if it has a dangling port that is required to be connected, and the port is highlighted in red when the user validates the workflow (the lower port of the Delete service in Figure 2 is such an example).

B. *Execute Perspective*

The execute perspective allows the user to run a workflow that they have composed. In most cases the user will just want to execute the workflow straight by simply clicking the "Run" button. However, SWiFT also supports more advanced execution modes. The user can set debugging *breakpoints* in the workflow to pause and resume during execution. Additionally, the user can set *viewpoints* where he or she wants to see any intermediate results of the workflow up until that point. The combination of these features allows for a powerful execution environment for more expert users.

Another feature that is valuable in the execute perspective is the use of *UI services*. Any parameters that are connected to a UI service will prompt the user for any necessary values while the workflow is executing. In Figure 2 we can see that the adjacency parameter of the Delete service is connected to th UI service. This means that, prior to execution of the Delete step in the workflow, the user will be queried for a value for this parameter. By using viewpoints and UI services appropriately, they can inspect the output and enter in the correct value during the workflow execution.

C. *Analyze Perspective*

The analyze perspective allows the user to more deeply examine their workflow beyond the rules of syntactic correctness

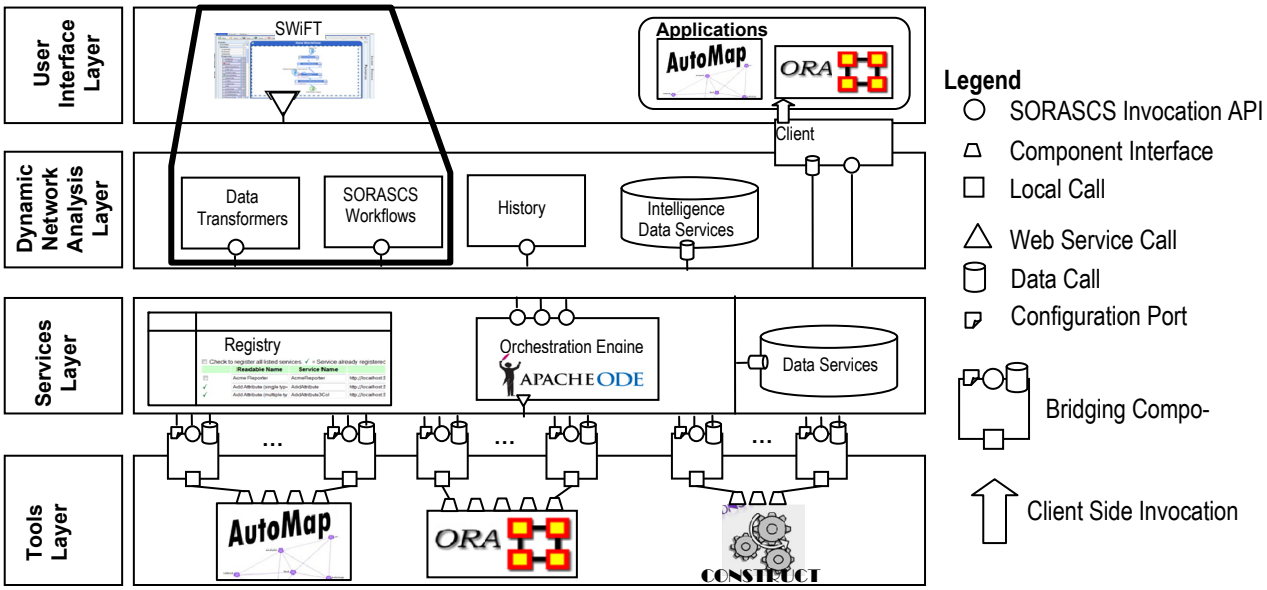

Figure 3. SORASCS System Organization.

described above. Using an extensible analysis plugin framework, the user has access to any public plugins that are available on SORASCS. These plugins have access to the current workflow, SORASCS services, execution history, etc. and can thus perform much more thorough and customizable analyses of workflows.

One such example plugin uses a simple bigram algorithm to make suggestions to the user based on existing workflows. For example, if the user has ordered service A before service B, but in most other workflows service B is before service A, the bigram analysis will suggest switching the order of the two services. It can also suggest a "next" service, based on the current services and the workflow history.

V. SWiFT Implementation and Engineering

SWiFT is built upon the SORASCS architecture, which provides services specific to dynamic network analysis. Figure 3 shows the SORASCS architecture, highlighting where SWiFT is positioned within it. The Tools layer of SORASCS contains the existing legacy systems that have been wrapped as services in SORASCS. The Services layer houses foundational services such as the Registry (for finding services), the orchestration engine (for executing workflows), and data services for accessing data stored in SORASCS. The Dynamic Network Analysis layer contains domain-specific services, including those used by SWiFT to compile and analyze workflows. The User layer contains user-facing parts of SORASCS, including the SWiFT front end.

The user interface for SWiFT is implemented as a web application using several JavaScript libraries, and uses AJAX for communication between the interface and the other layers of SORASCS on the SORASCS server.

There were several engineering challenges faced during design and implementation of SWiFT when implementing some of its requirements. Such challenges include providing appropriate levels of abstraction to bridge the gap between users and low level SOA concepts, how to provide sharing and reuse, balancing ease of use with version and state management, and how to provide for interactivity in a web environment.

A. Dynamic Network Analysis Layer

The first challenge that we needed to address was how to bridge the gap between human-centered abstractions for workflows and the lower level SOA concepts of orchestration and service calls. This challenge is addressed within SORASCS, by providing a domain-specific layer for dynamic network analysis. This layer contains models and services targeted specifically at the end user. One challenge in engineering this layer was how to provide abstractions that are formal enough to be used for analysis, guidance, and to be elaborated as orchestrations and service calls in the lower level, but still be amenable to end users. We have found that using component and connector architectural views at this level provides sufficient balance between these requirements. Workflows in SWiFT are mapped to architectural representations in the dynamic network analysis layer. A large array of architectural analysis can be performed on these models, including well-formedness checks, stylistic guidance, and quality of service analyses. The results of these analyses can then be mapped to the wiring model used in SWiFT. Additionally, the architectural models are formal enough that they can be used to generate BPEL scripts in the SOA layer, hiding the low level minutiae of forming service invocations, dealing with exceptions, and processing asynchronous calls. We believe that engineering any human centered platform for workflows will require a similar layer.

B. Sharing and Reuse of Workflows in SWiFT

One of the key requirements in SWiFT was for analysts to be able to reuse workflows created by other analysts and to incorporate them into other workflows. One consequence of this was the need to manage permissions of the workflows, as well as keep additional state in the GUI of opened workflows as users navigated between parent and child workflows. In

essence, there was a trade-off between achieving *usability* while hiding the workflow's version management and state management *complexity*.

First, SWiFT has an open workflows list in the palette of the GUI in the compose perspective. This allows the user to open several workflows at the same time so that the user can refer to other workflows during workflow orchestration. Also, SWiFT has three perspectives; compose, execute, and analyze. Users can move to any workflow tab or perspective, which requires state management of the workflow parameters on the GUI side and server side. We decided to keep all the information in GUI once after the workflow is opened, rather than accessing it from the server every time the user switches the tab or perspective. This improved the tool's responsiveness being on the web.

Second, SWiFT allows users to incorporate workflows within another workflow as one of its steps. Such a grouping is particularly helpful if a set of related activities occur repeatedly. SWiFT displays workflows in the palette, where they can be dropped onto the canvas in the same way as the other web services. SWiFT then allows the users to drill down to the internal workflow in any perspective to see its details on the canvas. Keeping the relationship of parent-child workflows and the list of open workflows in three perspectives was a challenging problem. Our initial attempt was to provide a generic solution. However, it led us to complexity that can be compared to supporting recursive types in programming languages. In order to manage the trade-off between the complexity of inter-workflow relationships and managing SWiFT's performance, we decided to enforce two constraints on the user's operation. First, the user cannot drill down an internal workflow unless the current workflow is saved. Also, the user needs to save the child workflow before he/she closes it, else the modification made to the child workflow will be lost. Second, we restricted opening multiple internal workflows at the same time. These constraints reduced the amount of information which SWiFT needed to keep on the GUI side.

Third, to tackle the issue of version management of the workflows as they are being shared with multiple users the following decisions were made. When a user is in execute or analyze perspective, the workflow is read only. Hence, the user is free to navigate between child and parent workflows in these two perspectives but cannot modify them. However, when the user is in the compose perspective, they may make changes to both children and parent workflows, depending on permissions. To get this right, we used a Z Specification [20][20] to help define the interface between the frontend and the backend as well as to understand the relationship between root/child/original workflow with infinite levels of workflow hierarchy.

C. User feedback and Interactivity

One of the engineering challenges for SWiFT was to support user interaction and to provide effective feedback to the user during various stages of workflow construction and execution. In this section we will describe how SWIFT handles this.

1) Feedback mechanism in SWiFT

SWiFT is built using Web 2.0 paradigm using AJAX technology. SWiFT's UI is based on single page model and is totally asynchronous when it comes to its interaction with middle tier. This makes the UI highly responsive. However this brings in the challenge of how to effectively communicate the feedback back to the user from the middle tier, for example during workflow execution and debugging. Web 2.0 doesn't provide a direct guidance for this, and leaves to the individual AJAX frameworks to provide a programming model. SWIFT uses Direct Web Remoting (DWR) [18] as its AJAX framework and effectively uses its Reverse AJAX programming model to send information back from middle tier to the GUI. Reverse AJAX [19] refers to an AJAX design pattern that uses long-lived HTTP connections to enable low-latency communication between a web server and a browser.

2) User Interaction in SWIFT

SWIFT's workflow execution engine is based on BPEL. User interactions are currently not supported by BPEL, which is primarily designed to support automated business processes based on Web services. However the spectrum of activities that makes up general purpose business processes is broader than this, because people often participate in the execution of business processes. To support a broad range of scenarios involving people within business processes, a BPEL extension is required. BPEL4People [6] has been a recent proposal to address this via a special purpose construct named WS-Human-Task, however many BPEL engines do not yet support it (including Apache ODE, SWiFT's BPEL engine). As mentioned in the requirements, DNA workflows supported by SWIFT are typically long running and can require user interaction at various steps during execution. Apache ODE lacks support for interactive workflows. Given this as well as its asynchronous nature as described above, it makes the task of supporting user interaction during workflow execution challenging for SWIFT. SWIFT attempts to solve this using existing BPEL constructs and using Reverse AJAX [19] in the following manner:

- User Interaction is modeled as a special type of utility service called as UI Service. This service can be hooked up to parameters of any other service in the workflow. SWIFT 's UI model has a type system to map service parameters to appropriate UI elements. This enables generation of a UI widget at runtime from a UI service based on the mapped parameters.
- The UI service is implemented as a stateless service in SWIFT 's middle tier. It has operations to check if the input is already provided by the user. During execution Apache ODE makes a call back to the UI service, which generates and pops up the UI to the end user (using the UI information captured as mentioned above). The UI is presented to the user by Reverse AJAX mechanism as described above. BPEL wait construct is used to poll the status of the UI service (using the operation of status check in UI Service mentioned above). We believe that this approach of handling works well for current workflows which are being developed using SWIFT. However this needs to be evaluated more, and needs more investigation into a task and inbox based approach, for example as is proposed in BPEL4People.

VI. DISCUSSION

Analysts conducting dynamic network analysis require a way to compose workflows that provides support for modeling heterogeneous, interactive workflows that are supported with special-purpose guidance, and can be shared and reused. In this paper, we describe a tool that supports such composition of workflows, called SWIFT.

Analysts can use SWIFT to combine various kinds of service components – ranging from web-services, tool APIs, data, and pre-existing workflows, where these are designed by multiple tool creators, running on a variety of distributed hosts. SWIFT also supports interactivity by not only allowing turnkey automation, where users can execute workflows as batch processes by providing required input parameters, but it also supports more complex workflows that need to ask the user for information dynamically as they get executed, and provides users visibility over intermediate results. This is extremely helpful for domains that have long running transactions, or have scenarios where users need to dynamically change parameters of workflows during their execution.

The plugin based architecture of SWIFT allows incorporation of various guidance mechanisms that can provide analyses for quality-attributes such as performance, security and execution time. These analyses help users to answer questions such as: `How long will it take for this workflow to execute?' or `Do all the services used in this workflow meet the security policies?'. This loosely coupled architecture not only allows easy incorporation of new analyses, but also ensures easy addition of new capabilities.

SWIFT supports the sharing and reuse of workflows and services by packaging and exporting them in a common repository along with their meta-information that are provided as tags.

Engineering SWiFT to address the requirements for this domain presented a number of key challenges that are likely to be encountered when designing workflow tools for other similar domains. We described how inserting a domain-specific layer that provides models and analyses that map between human-centered abstraction and implementation abstractions enabled us to provide tools that allow users to focus on their analysis tasks. We also described how the need for interactivity and reuse led to engineering challenges that required us to be careful to balance usability and engineering complexity.

Our evaluation of SWiFT has so far been informal. Yearly, as part of the development of SORASCS, we host community meetings where prototypes of SWiFT have been shown to about 50 people in the dynamic network analysis community, including tool developers and analysts. The feedback on this approach has been encouraging, and we look forward to more formal evaluations of our approach.

Future work on SWiFT includes providing additional forms of guidance such as automatic workflow repair, for example to automatically apply data transformations so users can be less concerned with data compatibility. Furthermore, we plan to provide additional analysis and advice concerning security and privacy issues with workflows, as well as staging advice for optimal performance in a distributed setting.

REFERENCES

[1] Carley, K. A Theory of Group Stability. American sociological Review, 56, 331-354, 1991.

[2] Carley, K.M., A Dynamic Network Approach to the Assessment of Terrorist Groups and the Impact of Alternative Courses of Action. In Visualizing Network Information Meeting Proceedings RTO-MP-IST-063, France, 2006.

[3] Carley, K.M., Reminga, J., Storrick, J., and Columbus, D., ORA User's Guide 2010. Carnegie Mellon University School of Computer Science Institute for Software Research Technical Report CMU-ISR-10-120, 2010.

[4] Borgatti, S., Evrett, M. and Freeman, L., UCINET 6 for Windows: Software for Social Network Analysis User's Guide. Analytic Technologies. 2002.

[5] Chris Peltz: Web Services Orchestration and Choreography. IEEE Computer 36(10): 46-52 (2003)

[6] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic. WS-BPEL Extension for People – BPEL4People. http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people

[7] Wei Tan, Paolo Missier, Ravi K. Madduri, Ian T. Foster: Building Scientific Workflow with Taverna and BPEL: A Comparative Study in caGrid. ICSOC Workshops 2008: 118-129

[8] Gorton, I., Wynne, A.S., Almquist, J.P., Chatterton, J. The MeDICi Integration Framework: A Platform for High Per-formance Data Streaming Applications. In Proc the 7th IEEE/IFIP Working Conference on Software Architecture, 2008.

[9] Department of Veterans Affairs. VistA – HealtheVet Mono-graph. 2008. http://www4.va.gov/VISTA_MONOGRAPH/docs/2008_2009_VistAHealtheVet_Monograph_FC_0309.pdf

[10] Bradley Schmerl, David Garlan, Vishal Dwivedi, Michael Bigrigg and Kathleen M. Carley. SORASCS: A Case Study in SOA-based Platform Design for Socio-Cultural Analysis. In Proceedings of the 33rd International Conference on Software Engineering, Hawaii, USA, 2011. In Press.

[11] Giardine B et al, Galaxy: a platform for interactive large-scale genome analysis. Genome Res. 2005;15:1451-1455.

[12] Hull D., Wolstencroft K., Stevens R., Goble C., Pocock M.R., Li P., Oinn T. Taverna: a tool for building and running workflows of services. Nucleic Acids Res. 2006;34:W729–W732.

[13] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J. and Zhao, Y. (2006), Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience, 18: 1039–1065. doi: 10.1002/cpe.994

[14] Goble CA, Bhagat J, Aleksejevs S, Cruickshank D, Michaelides D, Newman D, Borkum M, Bechhofer S, Roos M, Li P, De Roure D: myExperiment: a repository and social network for the sharing of bioinformatics workflows. Nucleic Acids Res 2010, 38:W677-682.

[15] Goble CA, Belhajjame K, Tanoh F, Bhagat J, Wolstencroft K, Stevens R, Nzuobontane E, McWilliam H, Laurent T, Lopez R: Biocatalogue: A Curated Web Service Registry for the Life Science Community. Microsoft eScience conference: 7-9 December 2008; Indianapolis

[16] Yahoo Pipes http://www.pipes.yahoo.com

[17] Carley, K.M., Columbus, D., DeReno, Bigrigg, M. and Kunkel, F. AutoMap User's Guide 2010. Carnegie Mellon University School of Computer Science Institute for Software Research Technical Report CMU-ISR-07-121, 2010.

[18] Direct Web Remoting. (2010). Retrieved from DWR - Easy Ajax for JAVA: http://directwebremoting.org/dwr/index.html

[19] Reverse Ajax. Retrieved from DWR - Easy Ajax for JAVA: http://directwebremoting.org/dwr/documentation/reverse-ajax/index.html

[20] SWIFT Drilldown Zed Specification - http://dogbert.mse.cs.cmu.edu/MSE2010/projects/faceo5/documents/Architecture/ArchitectureDocument/Drilldown%20Zed%20Spec/Workflow Spec%20%28cloning%29.pdf