End User Orchestrations*

Vishal Dwivedi, David Garlan, and Bradley Schmerl

Institute for Software Research
Carnegie Mellon University, Pittsburgh, PA-15213, USA
{vdwivedi, garlan, schmerl}@cs.cmu.edu

Abstract. Service-orchestrations define how services can be composed together and are widely used to execute applications based on Service Oriented Architectures (SOAs). However, the various special purpose orchestration languages used today require code-level constructs that force the users to provide excessive technical detail. For many SOA domains end-users of these orchestrations have limited technical expertise, and hence these users find it difficult to specify orchestrations in current languages. Additionally, users specifying orchestrations would often like to reason about architectural attributes such as performance, security and composability - capabilities that current languages and tools do not support. In this paper we provide an improved technique for modeling orchestrations that allows users to focus primarily on the functional composition of services that is guided by tool supported domain-specific analyses. We introduce an abstract architectural specification language called SCORE (Simple Compositional ORchestration for End users) that defines the vocabulary of elements that can be used in a service composition. SCORE not only allows users to create correct service-orchestrations, but it also removes the need for technical detail, most of which is auto-generated by tool support. We demonstrate the use of our approach to specify service-orchestrations in SORASCS (Service ORiented Architectures for Socio-Cultural Systems), which is a SOA system for the intelligence analysis domain. SORASCS users are analysts, who are involved with domain-specific analysis workflows that are represented using SCORE and executed.

1 Introduction

The raison d'être of web-services is that they can be composed with other services, data elements and middleware components to deliver composite functionality. Most SOAs are based on this principle and use orchestrations as an underlying method for specifying and executing such service compositions.

In early 2000, a number of open XML-based standards were introduced to enable service-orchestrations by international consortiums like OASIS, BPMI and the Apache Foundation. These efforts were led by large vendors like IBM, SAP and Oracle. The primary goal of these standards was to provide a uniform method of collaboration between orchestrated web-services by defining patterns at the message-level of service interaction. These standards were primarily designed for enterprise integration and provided the basis for current service-orchestration languages such as BPEL [1], BPML [2], and WSCI [3].

* Submitted for publication

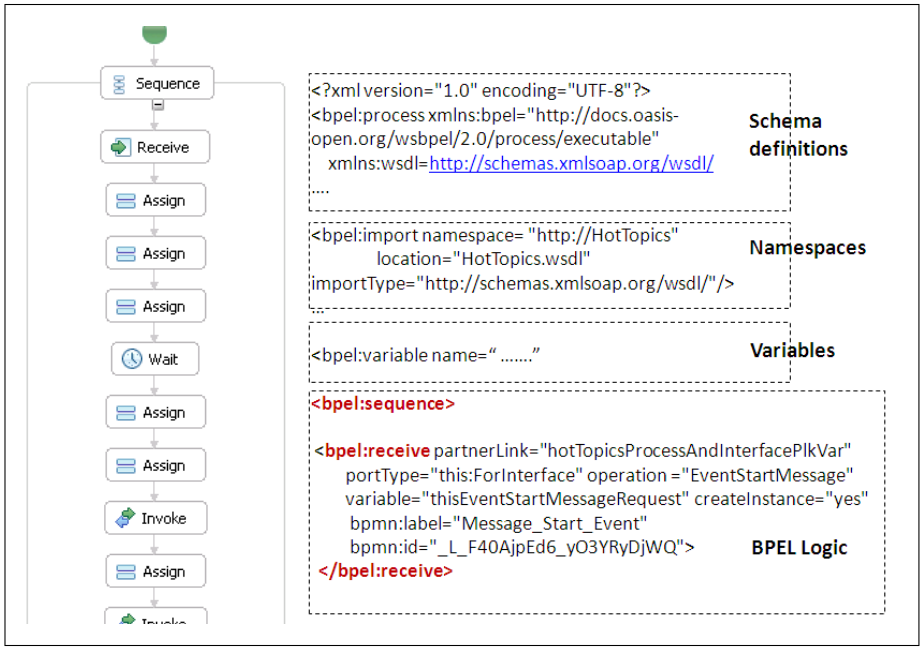


Figure 1: A snapshot of BPEL orchestration UI-elements and its XML code

Today SOA based systems are used across various domains where the computational model is aggregation of functions from various tools and services. In many cases, users of such orchestrations are domain experts, who are primarily concerned with composition of services, but have limited expertise to write detailed technical specifications. Languages like BPEL and BPML are not appropriate for this class of users as they involve code constructs that require descriptions of orchestrations at very detailed technical level. For instance, **Figure 1** depicts a simple service-orchestration segment showing the user-interface elements and the corresponding XML code using a standard BPEL-orchestration tool.

For a SOA domain where users are not computer savvy, understanding and writing such specifications is difficult because of:

1. **Complexity:** Existing languages require users to have knowledge of technical details such as schema-definitions, namespaces and variables, making it difficult for them to define orchestrations correctly and easily.
2. **Conceptual Mismatch:** Users think functionally (composition of activities in a workflow) while orchestrations involve the specifications of control flow and variables.
3. **Lack of analysis support:** Few mechanisms exist to reason about architectural drivers such as performance, security and composability, which can aid in the design of correct orchestrations.

Traditional service-orchestration languages require *developers* to write XML specifications, and deploy them, before they can be used by other users; but

that is hard for domains such as analytics that require dynamic orchestrations. Users in these domains need a modeling approach that allows them to focus primarily on high-level functional composition of services, and at the same time, allows them to create and execute correct service-orchestrations. Our work specifically addresses this class of users (whom we refer to as "end-users"). This paper proposes an improved approach for modeling service-orchestrations using an abstract architectural specification language called *SCORE* (Simple Compositional ORchestration for End users) that can be used to specify a high-level functional composition of services. SCORE provides a domain-specific vocabulary that can be tailored for different users and domains and does not require writing low-level code. Further, we provide various analyses to check composability, security and performance over SCORE specifications in order to help end-users to compose correct service compositions. These high-level SCORE specifications are automatically converted into executable specifications that can be executed by the system.

2 Distinguishing Characteristics of SCORE

SCORE can be characterized as providing an *abstract* language for specifying service-orchestrations, which can be *analyzed* for various quality attributes using a functional vocabulary and *executed* through tool support. We use this characterization to compare SCORE with the related work.

2.1 Abstract

SCORE provides an abstract component-oriented language that hides the technical details from end-users. It uses a functional vocabulary with a domain-specific component and property type system along with a rich set of constraints. This allows users to focus primarily on the functional composition of services. Orchestrations can be expressed in SCORE using a collection of styles that are designed for a particular SOA domain.

There exists similar work by Esfahani and colleagues at George Mason University, where they provide a language named SAS [4] for modeling functional and QOS requirements in an activity level specification. However, their approach relies on creating domain ontologies and mapping workflow activities to the domain. In contrast, SCORE uses a set of domain-specific architectural styles that can be customized for a specific SOA domain. Likewise, Mayer and Foster[5]used model-driven architecture (MDA) for orchestration code generation using a UML profile for SOA, but a single such profile is difficult to generalize across various SOA domains.

By comparison, most current service-orchestration languages are at opposite ends of a spectrum with respect to their support for abstraction. Languages like BPEL [1] and BPML [2] require specification of detailed code constructs like schema-definitions, namespaces and variable assignments and have minimal support for abstraction. Alternatively, UML-based languages like BPMN [6] allow abstraction at the level of composition of activities, but are used primarily for documentation purposes, and are not executable.

2.2 Analyzable

SCORE provides various analyses such as checking for composability, security and performance that aid end-users in composing correct orchestrations. Some of these analyses are based on domain-specific constraints that cannot be easily specified using unconstrained UML-based modeling approaches.

Although, there exists a considerable amount of work demonstrating analysis of workflows, current orchestration languages provide limited support for analyses beyond type-checking. One of the reasons for lack of analysis support can be attributed to the fact that most of the current approaches rely on modeling formal representations, and then using them to analyze structural or runtime properties. For instance Puhlmann et. al. proposed analyzing structural soundness of orchestrations [7]. Koshkina et al proposed checking concurrency of BPEL orchestrations by formally modeling BPEL in a process algebra [8]. Similarly, VanderAalst and colleagues have done work towards PetriNet-based analysis of workflows to check for control flow errors[9].

The fact that most current languages are built on XML with a relatively fixed schema, and allow limited support for adding additional attributes and constraints, makes it difficult to provide analysis at design time. Very often, the domain of the workflow enforces constraints that must be followed by end-users. SCORE provides support for adding properties and constraints, allowing designers to write domain-specific analysis that other languages find difficult to support.

2.3 Executable

SCORE specifications are compiled into low-level orchestration specifications that can be executed by an orchestration engine. Our current prototype associates the architectural specification with a script and uses the composition logic to generate low-level BPEL scripts. The generated BPEL scripts are deployed on Apache-ODE BPEL engine [10] and executed.

Whether a service composition language is executable or not depends on the level of abstraction it supports. BPMN [6] for example provides support for abstract specification, but is used primarily for documentation purposes. On the other hand, languages such as BPEL[1], BPML[2] and WSCI[3] are executable but are code-centric. There has been some support for executable BPEL code generation from BPMN models by vendors such as Intalio [11] but that is fairly restricted. In general, it is difficult to translate a free-form UML based diagram unless the constraints are codified - something which is much easier through architectural support for writing constraints.

3 Design Approach

We decided to design a two-layered specification for representing orchestrations, where each layer handles different concerns. The end-user specification language, named SCORE, is an abstract specification, is primarily functional in nature, and is used to design high-level workflows describing service compositions. Orchestrations are detailed executable specifications in a language like BPEL, which are auto-generated.

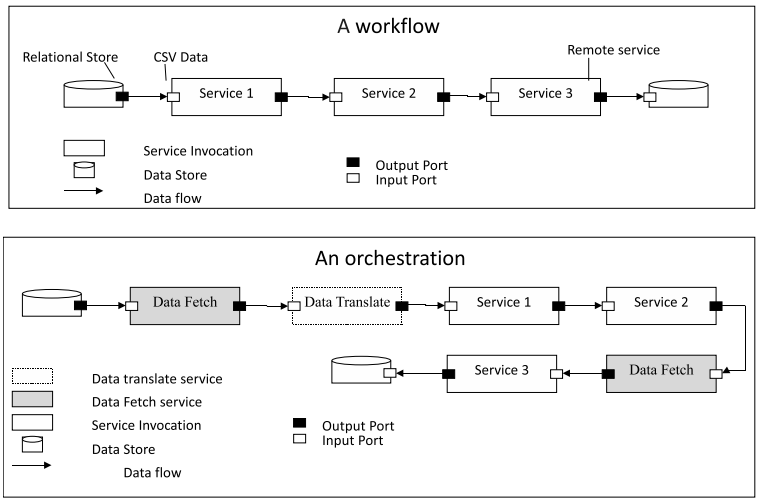


Figure 2: An illustrative two-layered specification

A simple example for such a multi-level specification is shown in **Figure 2**. The figure describes a simple workflow, which composes three services with varying input and output data requirements and location constraints. It also shows the corresponding orchestration that includes additional components for data translation and data fetching to compose a sound orchestration specification. Note that here DataTranslate element is representative of a large class of low-level components that must be present in an executable orchestration. For instance a realization of this translation service may involve additional operations such as conversion of relational tuples into XML, their XSLT based transformation, and conversion to a comma-separated file, each of which may require invoking additional services. Although, this is a simple illustration, it provides a glimpse of how abstract models can be helpful to end-users by providing just the necessary details allowing for a simpler end-user specification that can be translated into an executable specification using additional properties and constraints.

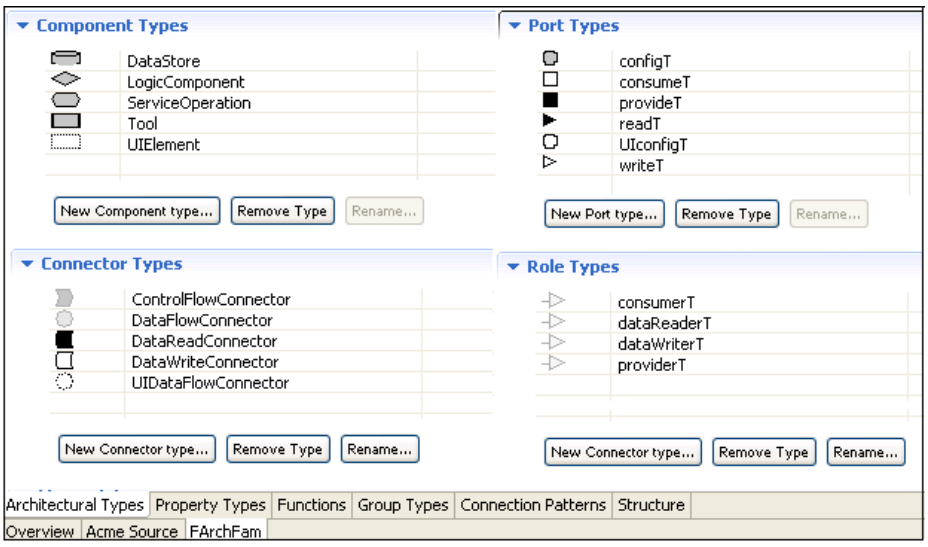
We provide a restricted vocabulary for the specification of workflows using SCORE, where it abstracts the specification of workflows to the essential component and connector types and the properties of concern. In this abstract vocabulary:

- *Components* represent the primary computational elements and data stores of a workflow.
- *Connectors* represent interactions among components that may vary from simple HTTP based calls to complex protocols
- *Properties* represent semantic information about the components
- *Constraints* represent restrictions on the service compositions that should remain true while designing orchestrations. Typical constraints include restrictions on allowable values of properties, composition restrictions, or membership constraints (as described in Section 3.2).

The above elements are provided via a predefined template (called a *style*) that defines the vocabulary of design elements for a particular domain. Unlike free-form UML based approaches for workflow composition, a style-centric approach constrains end-users to use the appropriate elements along with their associated restrictions. Such a style can be further customized or extened to include additional constraints based on the domain of usage, thereby providing more support to end-users. End-users who use the SCORE style to compose workflows are provided with default values for various attributes making their job more easier.

The next few sections describe the type-system of SCORE and how it can be used by end-users to generate orchestrations.

SCORE component types



SCORE Properties

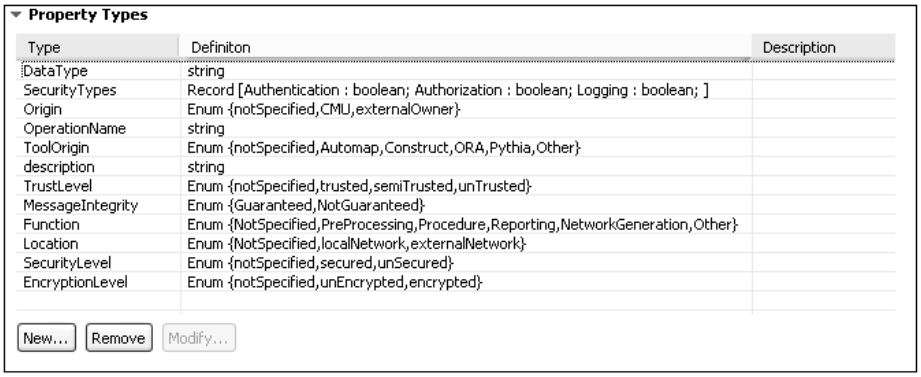


Figure 3: SCORE basic component and property types

3.1 Elementary vocabulary for SCORE

SCORE was prototyped using the Acme architecture definition language [12] via a special-purpose data-flow architectural style that provides (i) a component type system representing the SOA domain, (ii) a property system that can support analysis, and (iii) a set of analysis based on rules about properties of the workflow. In **Figure 3**, we detail the element types used in the SCORE vocabulary.

The high-level component types such as `ServiceOperation, DataStore, LogicComponent` and `Tool` are the elementary component types for the workflows and they represent web-services, data-access elements (such as file access and SQL data-access), control flow logic based on conditions and external tool APIs respectively. The `UIElement` is a high-level component type that handles user-interaction.

Similarly, the high-level connectors `ControlFlowConnector` and `DataFlowConnector`, as the names suggest, provide data-flow and control-flow semantics in a composition and are primarily responsible for communication between the components. The `DataReadConnector` and `DataWriteConnector` are primarily meant to read and write data from a `DataStore` Component. The `UIDataFlowConnector` is a special purpose AJAX connector that provides capabilities to interact with user interfaces.

The ports act as interfaces to the components and serve as interaction points for each component. The `consumeT` and `provideT` port types are responsible for representing data-output and data-input interfaces between the various components. The `readT` and `writeT` port-types are the data read and write interfaces for the storage components. The `configT` is a special purpose port type that serves as an interface to add configuration details to components. Similarly, `UIconfigT` port-type acts as an interface to the `UIElement`.

In this style, connectors rather than being unchangeably bound to specific ports, have interfaces that are defined by a set of roles. Each role of a connector defines a participant of the interaction represented by the connector. We define some primitive roles `consumerT` and `providerT` as service consumer provider roles and `dataReaderT` and `dataWriterT` as roles over `readT` and `writeT` ports. Each role of a connector defines a participant of the interaction represented by the connector. The ports and the roles in the SCORE style allow us to write various domain-specific constraints that restrict configurations of workflows, and thus avoid potential mismatches. Specifications written using these component types are then checked for consistency and well-formedness. We implement some of these consistency checks by enforcing constraints on the workflow elements. We would discuss these in detail in Section.

One of the critical trade-offs in the design of SCORE vocabulary was ensuring a balance between the complexity of the component type-system and the property type-system. They need to be expressive enough so that workflows in the domain can be represented using the provided vocabulary. However, since these workflow specifications form the basis of automated code generation, they have to capture enough detail to be able to do so.

In this section we described the type-system for the base style for SCORE. However, in practice, other sub-styles extend this base style to provide additional capabilities. These are customized for the domain they are used and are extended with additional domain-specific rules. We describe some of these domain-specific customizations in later sections.

3.2 Domain-specific constraints

`Workflow.Connectors`	The set of connectors in a workflow
`ConnectorName.Roles`	The set of the roles in connector ConnectorName
`self.PROPERTIES`	All the properties of a particular workflow element, where self is a pointer to the element itself
`size( )`	Size of a set of worklflow elements
`invariant`	A constraint that can never be violated
`heuristic`	A constraint that should be observed but can be selectively violated

Table 1: Sample Acme constructs

SL. No	Constraint type	Example
1	Structural	Checking that Connectors have only two roles attached `rule onlyTwoRoles = heuristic ! size(self.ROLES) = 2;`
2	Structural	Checking if a specific method of the service called exists `rule MatchingCalls = invariant forall request :! ServiceCallT in self.PORTS |exists response :! ServiceResponseT in self.PORTS |request.methodName == response.methodName;`
3	Property Constraints	Checking if all property values are filled in `rule allValues = invariant forall p in self.PROPERTIES| hasValue(p);`
4	Membership	Ensuring that a workflow contains only the 3 types of services `rule membership-rule = invariant forall e: Component in self.MEMBERS |declaresType(e,ServiceTypeA) OR declaresType(e,ServiceTypeB) OR declaresType(e,ServiceTypeB);`

Table 2: Sample constraints specified in predicate logic in SCORE style

An important role of the SCORE style description is to make explicit the preconditions that must be satisfied in order to create a well-formed workflow. In practice, not all workflow specifications are valid, and by enforcing rules that determine restrictions on the structure, property or membership of workflow, end-users are provided additional modeling support. The SCORE style specifies these rules that are evaluated at design time, enforcing certain restrictions on the kinds of services users can compose. Writing these constraints involves some degree of technical expertise, but these are associated with the architectural style, which is written once, and then used for modeling all workflows using the style. **Table 2** illustrates some examples of constraints represented in SCORE. The constraints in SCORE are based on Acme's first order predicate logic (FOPL),

where they are expressed as predicates over architectural properties of the workflow elements. The basic elements of the constraint language includes FOPL constructs such as conjunction, disjunction, implication and quantification. **Table 1** illustrates some functions and expressions provided by Acme, which are used to specify workflow constraints. These constraints are associated with various design elements of the SCORE specification such as the components, connectors, port or the entire workflow itself.

By comparison, current UML based languages utilize constraint languages like OCL [13] to check for well-formedness rules. SCORE's constraints uses a concrete syntax similar to OCL and can be associated with the workflow design-elements using FOPL based rules.

An important role of the SCORE style description is to define the meaning of syntactic constructs in terms of the semantic model of the orchestration. We achieve this, at least to a certain extent, by enforcing these domain-specific constraints. These not only prohibit end-users in creating inappropriate service compositions, but also promote soundness of the orchestrations by ensuring feedback mechanism via marking errors when a user fails any such constraint.

3.3 Support for analysis

As we discussed in Section 3.1, along with the component type system SCORE also provides a property type system. Some examples of properties include the specification of location information, tool origin of services, security credentials and other attributes over the elements of the SCORE style. In this section we describe some of the example analysis types that can be built using SCORE properties, such as analyzing a workflow for composability errors, checking for security or ensuring various performance constraints. The rules for these analyses are written as predicates which are analyzed for correctness.

Constraint type	Details
Data Integrity	Data-format of the output port of the previous connector matches the format of the input port
Semantic appropriateness	Membership constraints for having only limited component types are met
Structural soundness	All Structural constraints are met, and there are: * no dangling ports * no disconnected data elements

Table 3: Composability Analysis

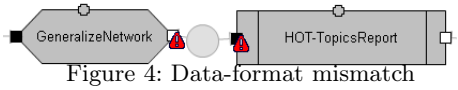


Figure 4: Data-format mismatch

Composability Analysis: Composability is defined as the ability to select and assemble appropriate components to satisfy a user requirement. An important aspect of composability analysis is to determine that the workflows designed by users are well-formed.

We ensure this by providing the analysis as shown in **Table 3**. Composability analysis is provided by a collection of rules expressed in SCORE. Workflows are created as instances of the SCORE style and all failures are visually marked as errors by the tool. **Figure 5** for instance illustrates a failure instance of a rule where an inappropriate method of service is called. The error is marked visually and displayed as an error message.

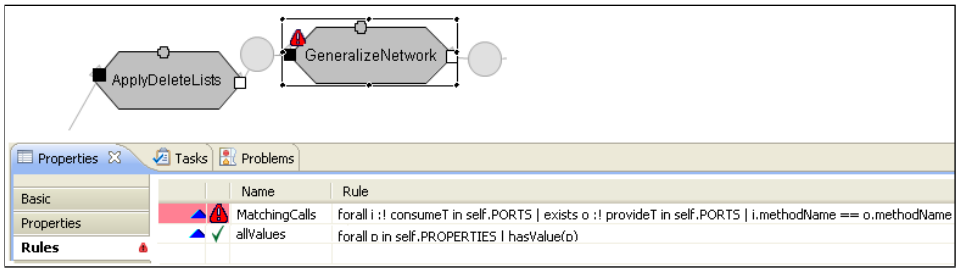


Figure 5: An error markup showing inappropriate method call

Influencing Factors		Requirements for security				
		Component Level			Connector Level	
Trust Boundaries		Authentication	Authorization	Logging	Encryption	Integrity
From	**To**					
Trusted	Trusted	✓	✓			
Trusted	Semi-Trusted	✓	✓			✓
Trusted	Untrusted	✓	✓			✓
Semi-trusted	Trusted	✓	✓			
Semi-trusted	Semi-trusted	✓	✓		✓	
Semi-trusted	Untrusted	✓	✓			✓
Untrusted	Trusted	✓	✓	✓		
Untrusted	Semi-trusted	✓	✓	✓		✓

Table 4: Security Analysis

Security Analysis: Another analysis provided by SCORE is to ensure that the modeled orchestration is secure. An illustrative example for this kind of analysis as shown in **Table 4**, which represents the security requirements for an organization. The table lists the security requirements for the various components and connectors for different modes of interactions. For example, if a trusted component invokes a semi-trusted component, the organizational policy requires the components to implement authorization and authentication, and ensure that the messaging mechanism assures integrity. A collection of such rules is represented in a style and workflows are analyzed to conform to the style. Note that the requirements for security may vary across domains and organizations. **Table 4** illustrates an example scenario of a security policy that needs to be enforced for all service compositions. SCORE can be used to specify such constraints, and analyze them.

Performance Analysis: SCORE can be used to analyze workflow performance by adding performance related properties to the components and connectors used to model the workflows and using an external plugin to perform the analysis. Some examples for such an analysis are : i) Evaluating approximate time to execute the orchestration, ii) analyzing that the orchestration would not stall (for example due to cycles), etc.

There exists a considerable amount of literature for analyzing workflow performance varying from stochastic analysis based on queuing theory, PetriNet based modeling and other formal approaches. SCORE provides an architectural style which can form the basis of such analyses, each of which can be implemented as plugins implementing the individual analysis code.

4 SCORE use-case

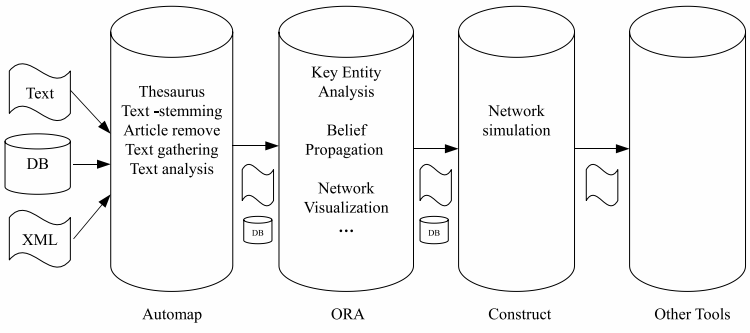


Figure 6: SORASCS tool pipeline

We use SCORE as an underlying style to represent workflows from SORASCS (Service ORiented Architectures for Socio-Cultural Systems) [14]. SORASCS is an end-user SOA system and an architecture platform for the integration of a set of intelligence analysis tools and services from Carnegie Mellon and other partner institutions. The computational model in SORASCS is based on aggregation of services and functions from various analytical tools and web-services from different organizations. Users in this domain are intelligence analysts who have expertise in analyzing data, but use computers merely as tools to accomplish their tasks; their computer expertise is limited to only general use and training in programs from their domain.

Intelligence analysts have a wide variety of computer programs that assist them in their tasks. **Figure 6** shows a set of such tools developed at the Center for Computational Analysis of Social and Organizational Systems (CASOS) at Carnegie Mellon University: AutoMap [15] for extracting networks from natural language texts, ORA [16] for analyzing the extracted networks, Construct [17] for what-if reasoning about the networks. In addition to these tools, there are a variety of tools from other organizations that may contain overlapping functionality.

While the existing tools were quite powerful in terms of functionality, they are mostly standalone tools that require training to use, and are difficult to

integrate and use together. There is a need in the community to mix and match the capabilities of many tools, depending on need, to record common sequences of usage for particular cases, to provide traceability so that analysts can determine how conclusions were reached by tools, and to be able to assemble multiple tool functions, data sources and new capabilities. Such needs can be addressed by providing tool functionality as services in a Service Oriented Architecture (SOA), that supports end-user workflow construction.

SCORE was used as an underlying architecture style for SORASCS as it provided capabilities for composing SORASCS services into workflows. The tool capabilities were thus decomposed into fine grained web-services that allows them to be orchestrated together to perform a multitude of tasks. While almost all services provided by SORASCS are currently fine-grained web-services implemented by wrapping the functionality of tools, the granularity of these services varied from very fine grained text-processing data-services in Automap, to bulky application services in ORA visualization. SCORE allowed support for orchestrating such thin-client and thick-client services. Although, creating an architectural style for the domain involved an additional overhead, but it was a one time job. Once defined, SCORE provided considerable support to end-users to model workflows in SORASCS and generate the corresponding executable orchestrations.

4.1 SORASCS Workflows in SCORE

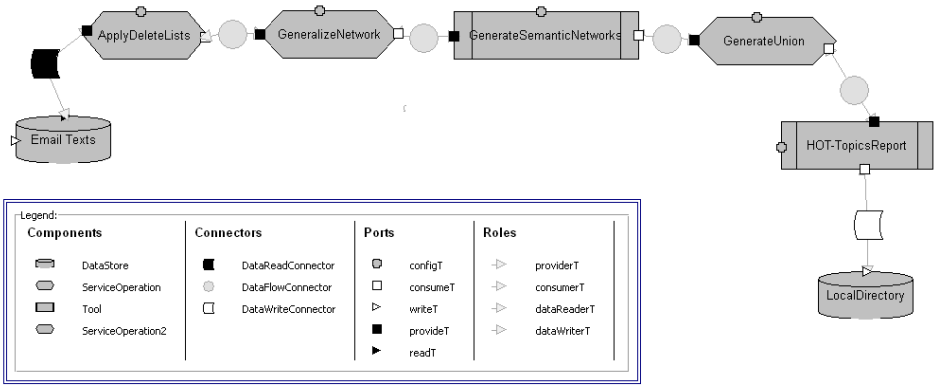


Figure 7: HotTopics Workflow composed using SCORE

Figure 7 provides a simple illustration of a workflow composed using SCORE. The workflow describes a composition of few domain-specific web-services, a visualizer tool and input and output data elements. This scenario from the SORASCS system [14] describes the *HotTopicsWorkflow* which involves extraction of data from emails, generation of a network representation, visualization of the network and creation of a report which describes the key actors who are involved in the text description. The workflow uses 5 key services - ApplyDeleteList, GeneralizeNetwork, GenerateSemanticNetworks, GenerateUnion, and HotTopicsReport. These services primarily deal with network extraction

and generation tasks for the intelligence analysis domain and are hosted on the SORASCS platform. `ApplyDeleteList` and `GeneralizeNetwork` services are responsible for deleting some text key-words and performing natural language processing on the text. `GenerateSemanticNetwork` service creates a set of networks for analysis that are combined together by the `GenerateUnion` service. These are then fed to `HotTopicsReport` service - a thick client based service to generate a report.

4.2 Code generation

SCORE specifications are designed to be abstract in nature as they enable functional composition of services based on a given domain-vocabulary. We use these specifications to auto-generate low-level orchestration code. Our code generation approach primarily consists of associating chunks of BPEL code with SCORE components, and using these specifications to create executable BPEL scripts. The generated BPEL scripts are then deployed on ODE BPEL engine[10] and executed. We tested our approach for defining SORASCS workflows, where the standard size of BPEL orchestrations varied from 100 lines to 4200 lines of code. It was obvious that analysts would find it difficult to model orchestration of this large size. SCORE not only automated the code generation, but it also made it easier for the end-users to model them by providing an abstract and functional vocabulary.

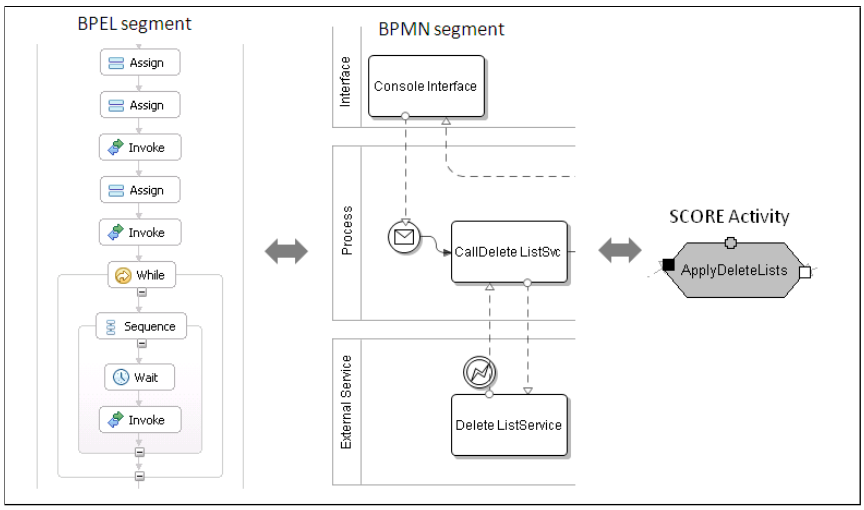


Figure 8: Comparison of constructs of the HotTopics workflow modeled in BPEL, BPMN and SCORE

5 SCORE concepts evaluation

In Section 2 we identified the distinguishable characteristics of SCORE. We also discussed how an abstract, domain-specific and functional vocabulary enables SCORE to provide end-user representation. This section describes how SCORE meets the criteria we identified earlier.

We designed a small experiment to compare the concepts required by current orchestration languages with the ones by SCORE. We use the SORASCS intelligence analysis domain as a standard domain for our test. Specifically, we use the same HotTopics Workflow described in **Figure 7** and model it using BPEL, BPMN and SCORE. Choreography languages like WSCI were not considered suitable for end-users in the SORASCS domain as they involve writing message interchange specifications in XML that are not easy to specify for users with limited technical expertise.

Type	**BPEL**	**BPMN**	**SCORE**	**WSCI**
Abstract representation	-	+	+	-
Functional vocabulary	-	+/-	+	-
Domain-specific constraints	-	-	+	-
Support for analysis	-	-	+	-
Executable	+	+/-	+	+

Table 5: Comparison of modeling support

NR*: Not Required

Concepts (by type)	**BPEL**	**BPMN**	**SCORE**
No. of Activities	70	13	7
No. of Variable Assignments	14	14	NR*
No. of Correlation Parameters	5	5	NR*
No. of Messages	2	NR*	NR*
NameSpaces	Required	NR*	NR*

Table 6: Comparison of Concepts required to model the HotTopicsWorkflow

Table 6 provides a snapshot of the concepts involved in creating such an orchestration using BPEL, BPMN and SCORE. We provide a logical comparison of the concepts involved in modeling the same workflow. Languages like BPMN and BPEL make it hard to model such orchestrations for end-users due to the complexity of the technical constructs involved. **Figure 8** provides an activity-level comparison for modeling the same HotTopics workflow. It is noticeable that it's not so much the number of activities, but the more technical vocabulary of the domain that allows significant reduction in the code-level semantics. BPMN provides a certain level of abstraction, but is not executable. Languages like BPEL and BPMN also do not provide any support to add constraints and properties to perform architectural analysis. By comparison, SCORE requires lesser number of concepts, making it easier for end-users to define orchestrations.

6 Conclusions and Future Work

In this paper we describe SCORE, an abstract architectural specification language for modeling service-orchestrations. Our work is inspired by the problems end-users face in modeling orchestrations and the limited analysis capabilities of the current orchestration languages. SCORE not only allows simpler modeling capabilities, but it also provides additional support for design time analysis. Our aim is to address the requirements of end-users who are

primarily concerned with service composition and analysis, but have limited technical expertise to write code. SCORE addresses such users by providing predefined templates representing the vocabulary and rules that can be customized for different domains. This domain-specific vocabulary is further supported by additional pre-built analyses for various quality attributes.

SCORE is part of our ongoing research effort in building SORASCS, an end-user SOA system for the intelligence analysis domain. We are currently working on an improved front-end to simplify service-orchestrations using SCORE. In the near future, we would like to provide various domain-specific sub-styles and other analysis that can allow easier composition of services by end-users. We believe that an architectural approach to represent service compositions can be extended to various other composition scenarios, providing more formal reasoning support for various quality attributes.

7 Acknowledgements

This work was supported in part by the Office of Naval Research (ONR-N000140811223). Additional support was provided by the Center for Computational Analysis of Social and Organizational Systems (CASOS). The views and conclusions contained in this document 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.

References

1. BPEL: BPEL Web Services Business Process Execution Language. (http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html/)
2. BPML: Business Process Modeling Language. (http://xml.coverpages.org/bpml.html)
3. WSCI: Web Service Choreography Interface. (http://www.w3.org/TR/wsci)
4. Esfahani, N., Malek, S., Sousa, J.P., Gomaa, H., Menascé, D.A.: A modeling language for activity-oriented composition of service-oriented software systems. In: MoDELS. (2009) 591–605
5. Mayer, P., Schroeder, A., Koch, N.: Mdd4soa: Model-driven service orchestration. In: EDOC. (2008) 203–212
6. BPMN: Business Process Modeling Notation. (http://www.bpmn.org/)
7. Puhlmann, F., Weske, M.: Interaction soundness for service orchestrations. In: ICSOC. (2006) 302–313
8. Koshkina, M., van Breugel, F.: Modelling and verifying web service orchestration by means of the concurrency workbench. ACM SIGSOFT Software Engineering Notes **29**(5) (2004) 1–10
9. Aalst, W.M.P.V.: Workflow verification: Finding control-flow errors using petri-net-based techniques. In: Business Process Management. (2000) 161–183
10. ODE: Apache Orchestration Director Engine. (http://ode.apache.org/index.html)
11. Intalio|works: BPMS. (www.intalioworks.com/products/bpm)
12. Garlan, D., Monroe, R.T., Wile, D.: Acme: An architecture description interchange language. In: Proceedings of CASCON'97, Toronto, Ontario (1997) 169–183
13. OCL: Object Constraint Language. (http://www-st.inf.tu-dresden.de/ocl/)

14. Garlan, D., Carley, K.M., Schmerl, B., Bigrigg, M., Celiku, O.: Using service-oriented architectures for socio-cultural analysis. In: Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE2009), Boston, USA (2009)
15. Diesner, J., Carley, K.: Automap1.2 - extract, analyze, represent, and compare mental models from texts. Technical Report CMU-ISR-07-114, Carnegie Mellon University (2004)
16. Carley, K., Reminga, J.: Ora: Organization risk analyzer. Technical Report CMU-ISRI-04-101, Carnegie Mellon University (2004)
17. Schreiber, C., Singh, S., Carley, K.: Construct: A multi-agent network model for the co-evolution of agents and socio-cultural environments. Technical Report CMU-ISRI-04-109, Carnegie Mellon University (2004)