

Semantic Interoperability Scripting and Measurements

Ke-Thia Yao, In-Young Ko, Robert Neches, Robert MacGregor *

USC Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292, U.S.A.
{KYao, IKo, RNeches, MacGregor}@isi.edu

Abstract

Assembling software from components is a long-sought vision. However, it is still difficult to insert/replace components in complex software systems and to ensure compatibility. We propose a set of mechanisms that directly address the issues of adaptive composition sensitive to quality concerns. Our approach helps software developers engage in guided, efficient searches and gauge-based evaluations of the set of alternative system implementations that can be built with the components available to them. Our tools support intertwining of composition and manual programming to iteratively build adapters for fitting components into a system when they are functionally satisfactory but suffer interface mismatches. Semantic Interoperability Measures Template Based Assurance of Semantic Interoperability in Software Composition (SIM TBASSCO) helps designers and developers understand the tradeoffs of alternative implementations, and uses records of decisions to generate run-time monitors that warn when the resulting system is being pushed outside its design envelope.

1. INTRODUCTION

1.1 Component Interoperability Problem

Assembling software from components is a long-sought vision. However, it is still difficult to insert/replace components in complex software systems and to ensure compatibility.

Component frameworks, such as CORBA and DCOM, and Architecture Description Languages (ADLs) help by capturing information at the component interface level that bears on compatibility. But, what they provide is both stricter than necessary and less than sufficient. Current tools focus on exactly matching required vs. available input/output data types and behavior descriptions. This is necessary for plug-compatible substitutions, but too restrictive for adaptation of closely related components. Current approaches capture interface and method

specifications, but not implementation factors. This supports composition up to a point, but is not sufficient to address qualitative considerations in composition, such as implementation effort, performance, resource requirements, or reliability.

We propose a set of mechanisms that directly address the issues of *adaptive composition sensitive to quality concerns*. Our approach helps software developers engage in *guided, efficient searches and gauge-based evaluations* of the set of alternative system implementations that can be built with the components available to them. Our tools support intertwining of composition and manual programming to iteratively build *adapters* for fitting components into a system when they are functionally satisfactory but suffer interface mismatches. Semantic Interoperability Measures Template Based Assurance of Semantic Interoperability in Software Composition (SIM TBASSCO) helps designers and developers understand the tradeoffs of alternative implementations, and uses records of decisions to generate run-time monitors that warn when the resulting system is pushed beyond its design envelope.

1.2 GeoWorlds Test Bed

The system application that we use to test our efforts is *GeoWorlds*, a component-based information management and analysis system capable of handling both geographic and Web-based information. *GeoWorlds* combines Geographic Information Systems software with tools for finding, filtering, sorting, characterizing, and analyzing collections of documents accessed via the Web [20, 6, 7, 13, 25]. *GeoWorlds* is currently in experimental use by the Crisis Operations Planning Team at the US Pacific Command (PACOM), and by the Virtual Information Center, also at US Pacific Command. Example uses at PACOM include: (1) mapping terrorist bombings in the Philippines; (2) locating patterns of recurring natural and technological disasters in China and India; and (3) investigating drug trafficking and piracy in various locales.

* Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreements F30602-00-2-0610. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DARPA, the Air Force Research Laboratory, or the U.S. Government.

2. APPROACH

Our approach centers on tools for developing *abstract system scripts*, which define a framework for exploring alternative system implementations by drawing from among candidate component sets for each function delineated in the abstract system. *Semantic component descriptions* go beyond component interfaces (data types, such as integers and floats) by enabling comparisons of functional compatibility and data equivalence, and thereby providing better assurance of compatibility. This is enhanced by a language of *qualifiers* on component software descriptions that supports qualitative evaluation during the composition process. Formal architectural level views of system execution (deployment, sequence and activity diagrams) provide easier ways to calculate resource usage and analyze performance.

Our approach to software component interoperability is based on three key ideas:

- **Metadata-level modeling of software components** to describe their behavior and input/output requirements at a more abstract level. Modeling at this more abstract level allows us to reason (inference) about interoperability among components connected to each other, and about compatibility among components performing similar functions. This frees us from the overly strict requirement of traditional typing mechanisms that enforce compatibility by requiring components to implement that same interface, or to inherit from the same class.
- **A metadata-level and system-level scripting mechanism** that describes the execution behavior of components and data-flow among components. The metadata-level scripting allows the application developer to create component-based applications that are guaranteed to make sense at the more abstract behavior and requirements level. In addition, by instantiating from metadata-level to the system-level the scripting mechanism can run a script by invoking the individual components and by providing the data-flow mechanism needed to move data from the output of one component to

the input of another component.

- **Software gauges that measure between-component interoperability and compatibility levels** (as opposed to a Boolean value). These software gauges are useful during both system design, which is the focus of this paper, and during system development. During the design phase, these gauges aid: selection by application developers of the most suitable components to use in scripts; evaluation by component developers of how well their products integrate with existing components; and installation by system administrators of the best system-level instantiations of scripts.

3. METADATA LEVEL MODELING

A vision of component-based software development is to provide a repository of software components for application developers to select from to rapidly build their product. In order for this vision to work the application developers must have confidence that the selected components are meaningfully interoperable. Component interface specifications, such as IDL, do not capture enough information to ensure that the semantic intents are satisfied during software composition. Here we propose a metadata level model to capture and reason about this semantics.

The insufficiency of interface-based specification is especially noticeable in the case of GeoWorlds, where a uniform service component API is adopted, and a common document collection data structure is used for most of GeoWorlds' information management service components. Within GeoWorlds many services can be forced syntactically together with no compilation error. Although this provides a very flexible mechanism to combine services, when used incorrectly it may generate run-time exceptions or strange results.

3.1 Content and structure forms

We have adopted a lightweight, multi-form ontology to present the semantics of document collections. This model is designed to capture essential aspects of

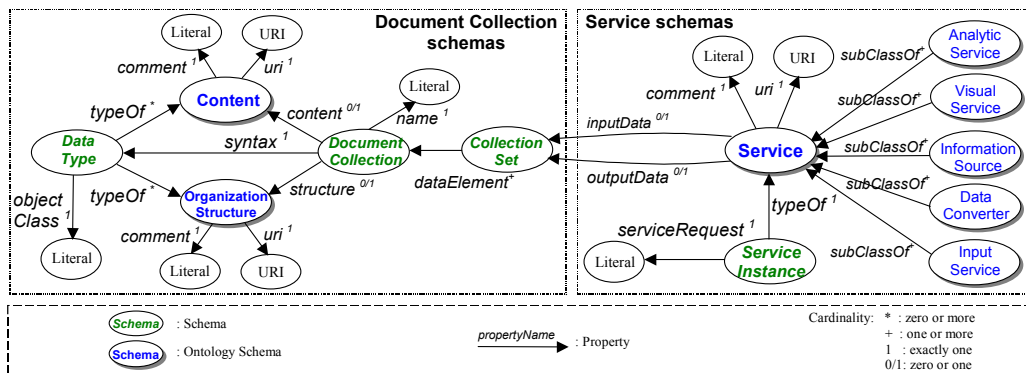


Figure 1: Schema model for representing document collection and service semantics

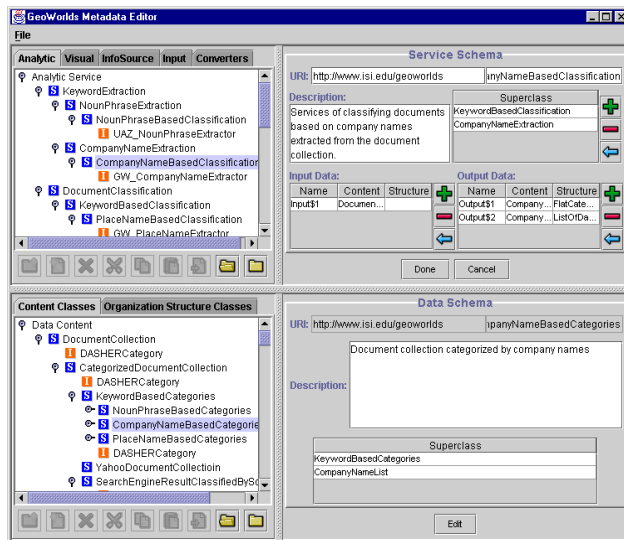


Figure 2: Metadata Editor

components that are of interest to the developer, while omitting semantic forms that would prevent the system from performing quick inference. Based on our experience with GeoWorlds, this lightweight representation is sufficient for most *current* Web-based information management services.

The semantic specification of a document collection can be divided into two forms: *content* and *structure*. The *content description* represents the contextual meaning of the collection (e.g., a document collection in which the documents are classified by the major noun phrases). The *structure description* characterizes the organization structure (e.g., a document collection organized in an acyclic graph structure).

Domain-specific ontologies are used to discriminate and classify the document collection content types and organization structures, and service functionalities. A schema model has been developed to represent the document collection and service semantics in terms of its content, organization structure, and active relations between other document collections. Figure 1 illustrates this schema model as an ER (Entity-Relationship) diagram. Content, Organization Structure, and Service are the top-level ontology schemas that describe the top concepts in the ontology hierarchies of instantiated schemas, which we call nodes. Subsumption relations between nodes can be described by subClassOf property. In the current implementation, Analytic Service, Visual Service, Information Source, Data Converter, and Input Service are defined as the major service types that are subclasses of Service. For more details of the metadata level modeling and reasoning mechanism using the model see [12].

3.2 Metadata Editor

The metadata editor is the system's GUI for registering and modifying semantic descriptions of document collections and services. By using this tool, local resource instances can be classified under the ontology hierarchies and their syntactic descriptions can be edited. Figure 2 shows a screen shot of the metadata editor. The upper part of the window is for editing service nodes (analytic, visual, information source, user input, and data conversion services). The lower part is for editing content and structure semantics of document collections. In both cases, the left side displays the ontology hierarchy and the right side provides forms to edit node property values¹.

4. SEMANTICALLY-BASED SERVICE SCRIPTING

The *Service Scripting Tool* [12] allows application developers and end users to combine multiple services together to perform complex information analyses. This tool provides a GUI for visually composing a service script by means of a data flow diagram. During the service scripting time, this tool uses the metadata level model to ensure that users only select interoperable services to create semantically well-formed scripts. Scripts can either be concrete (specifying specific services to use) or abstract (using metadata to specify classes of services to use). Abstract scripts can be dynamically instantiated at run-time based on components available from local component repositories. This dynamic instantiation not only allows for adaptation of the script to system environment changes, but also it allows for adaptation to utilize components newly added to component repositories.

4.1 Model for representing active document collection scripts

The scripting tool requires semantic descriptions that model the behavior of components it operates on. The model previously described for representing semantics of document collections and services has been extended to provide a model to represent active document collection scripts and their instances. Figure 3 shows the ER diagram of this extended model.

When an information service is matched against a set of document collections, the collections within the set should be bound to the service as marshaled input parameters, so that the service can be performed by using

¹ In the example shown, the upper-left part currently shows the ontology hierarchy of analytic services. The upper-right part shows the node properties for one of these services: "Company Name Based Classification." The lower part displays the outputs currently being edited in that service description, in this case, the ontology hierarchy for the document collection contents, and the node properties for "Company Name Based Category."

the data at run-time. Also, when a service is combined (pipelined) with another service, the output document collections of the first service should be bound to the input parameters of the second service. *marshaledInput* and *marshaledOutput* properties of Service schema represent such data bindings between nodes. Using Collection Set Instance schema, which elements are Document Collection Instance's, can represent a set of document-collection instances. An *analysisResultOf* property in a Collection Set Instance explicitly represents the input-output relationship between the set and another document-collection set via a service. Each Document Collection Instance has an *object* property which points to the physical object that keeps the content of the document collection.

A document collection can be a member of multiple I/O data sets and it can participate within multiple document relations. For example, consider a document collection composed of Spanish documents and a document collection categorized based on place names cited in the document contents. These document collections are the results of an English-to-Spanish translation service and a place name extraction service performed on an initial document collection.

Figure 4 shows an example of an active document collection script composed for organizing an information space on “High-Speed Internet Coverage Areas in the US.” By using the model explained at Figure 3, requirements on document collections and active relations between them are described. In this script, an initial collection obtained by merging string searches is analyzed in various ways (e.g., grouping by frequently occurring phrases, plotting location references on maps) to help identify and populating a topic hierarchy that organizes the collection.

Based on the active semantic relations between document collections, data flow between the services can

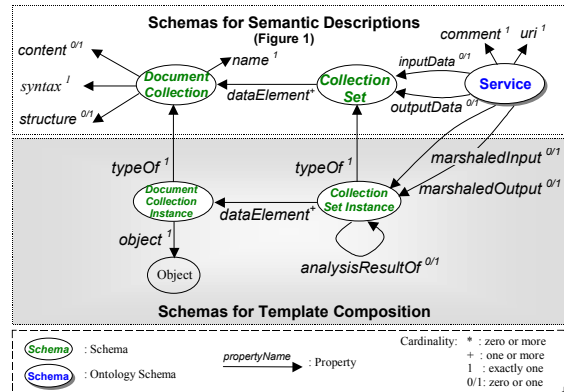


Figure 3: Schema model for representing active document collection scripts

be deduced (the shaded arrows in Figure 4 represent the data flow). For example, since the service f_4 requires d_1 and d_2 as its inputs, the data flows from the services f_1 and f_2 which are the producers of d_1 and d_2 , respectively to the service f_4 . Current implementation of the script composition tool allows users to compose an active document collection script by specifying data flow between information management services. The detailed input/output relations between document-collection sets and services, and *resultOf* relations between document-collection sets are automatically generated by the composer based on the data flow specified by the user. This makes the script composition task easier for the users.

The graphic representation of an active document collection script, shown in Figure 4, can be serialized (the current version of the system generates XML data). This can be stored in a script repository or exchanged with other users.

4.2 Script Instantiation and Execution

An active document collection script can be

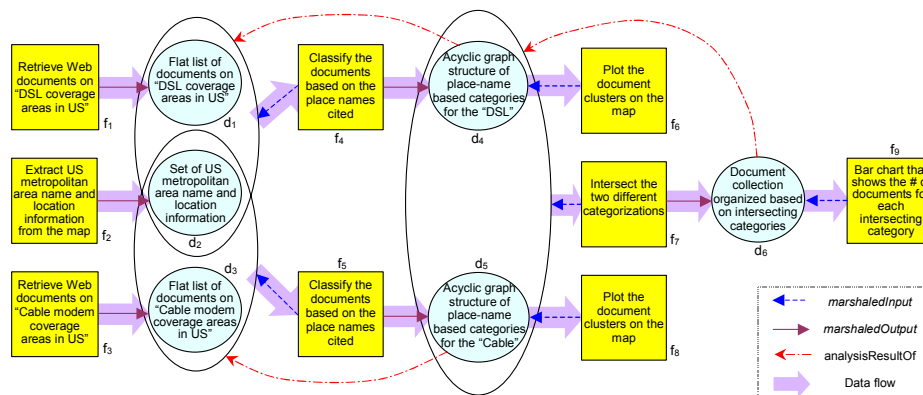


Figure 4: Active Document Collection Script for “High-Speed Internet Coverage Areas in the US”

instantiated by allocating local resources to the nodes in the script. In a local system, semantic descriptions about the local resource instances are kept as metadata in a repository. Metadata about a resource instance also includes some syntactic information such as data types, job request entries, and I/O parameter ordering². The semantic compatibility measurements (explained at Section 5) are performed to select semantically compatible local resources for each node in the script. The process of instantiating a script may require human interaction to resolve multiple matches of resource instances and syntactic mismatches between nodes³.

As the result of an instantiation, proxies are created to act as clients to invoke the specific services that are selected to instantiate the script, and to receive the results of these service instances. Each *proxy* delegates a resource instance (a document collection or a service) and keeps information for accessing the local resource. For each functional semantics, a service proxy is created and for each document-collection set, an ordered-list (ordered based on the I/O parameters of the corresponding service instance) of document collection proxies is created. A service proxy keeps precedence relationship between predecessor and successor proxies and a pointer to the corresponding semantic description in the script. A document collection proxy maintains a pointer to a document collection object.

The instantiated script can be executed by running the service proxies in a sequence governed by the precedence relations. An activated service proxy submits a service request to the system interface, monitors the job status, and receives the result. In the current prototype implementation, this service access mechanism is implemented based on the GeoWorlds' asynchronous service invocation architecture, which is described in [26].

Multiple service proxies can be run in parallel and the proxies can be synchronized by using the precedence relations. When a proxy receives the result from its service instance, it updates the object pointer fields in its output document collection proxies to point to the result objects. Then, it invokes all the successor service proxies. A service proxy will not be activated unless it receives signals from all the predecessors. Usually the terminal services are visualization services that have no successor nodes.

Active document collection scripts for complex information management tasks can be developed

² In the current prototype, Java class names are used to describe the internal data types of document collections. A job request entry is a directive to request for a service.

³ The current system resolves some syntactic mismatches automatically by finding and inserting syntactic converters.

incrementally by repeating the composition, instantiation and execution steps. After/while executing a script, the user can modify the script by adding new nodes or removing unnecessary nodes or replacing certain nodes with other semantically compatible ones (to specialize, generalize or alter the functionality). When a script node is modified, only the proxies that are affected are regenerated (usually from the modification point to the terminals). Therefore, when the script is re-executed, the unaffected proxies will not be rerun and all the intermediate data can be reused for the newly initiated services. This incremental development of active document collection scripts is especially effective when the information management task has to deal with large document collections and is composed of many analysis steps.

4.3 Service Scripting Tool

The active document collection script composer provides GUI-based tools to help users set up analysis and structuring activities by composing, instantiating and executing scripts. Figure 5 (a) shows the script composer window. The upper part displays the current script including services and connections between them. Given a selection of nodes in the script, the lower part shows the partial ontology hierarchies that are semantically compatible with the selected nodes. This lets users see what options are available to them for adding steps to their analysis.

As described in the previous section, the composer creates proxies and proxy connections (a directed acyclic graph of control and data flow between proxies) when a script is instantiated. This enables the system to invoke, monitor and synchronize the services. Also, with the help of the inference engine, it automatically finds and inserts syntactic converters (e.g., data type converters) between syntactically mismatched components if appropriate converters are available. When a script is executed, each node in the graph displays the status (progress bar and messages) of the service.

5. SOFTWARE GAUGES

The metadata level modeling and the service scripting mechanism described in the previous sections provide the underpinnings for a functional software component repository. Here we describe a set of software gauges that facilitates the usage and maintenance of the repository. Our current work supports gauges geared towards three types of repository users:

- *Interoperability gauges* for application developers to efficiently search for candidate components in the script generation process.
- *Compatibility gauges* for system administrators to adapt scripts to their local system environments by

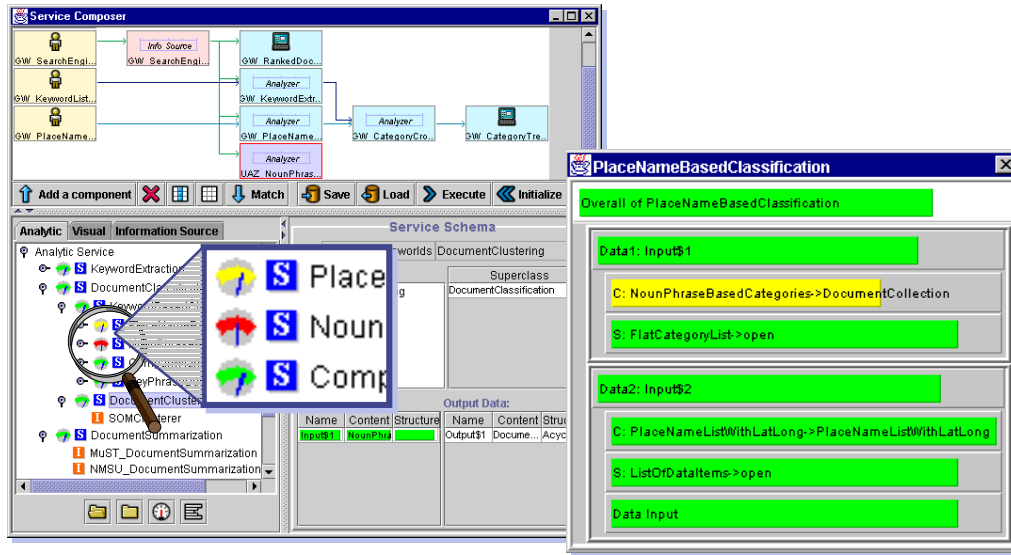


Figure 5: (a) Service Scripting Tool; (b) Compound Interoperability Gauge

finding candidate replacement components.

- *Insertion gauges* for component developers to judge the level of effort required to insert their component into the repository, and determine the uniqueness of their component within the repository.

Currently, the main metric the software gauges use is the semantic distance based on graph distance in the representation we use to model components. Given two data schemas in a taxonomy hierarchy, this metric determines if one schema subsumes (is the ancestor of) the other schema. If it does subsume, it determines the graph distance of the two schemas in terms of their content and structure. Finally the metric returns a value inversely proportional to the graph distance.

5.1 Interoperability Gauge

The Service Scripting Tool allows users to script semantically correct service data flow diagrams. At each step of the service construction process, the tool offers a set of semantically interoperable services for the user to choose. However, for the inexperienced user the number of choices can be confusing and overwhelming. To help a user to make appropriate selection we have implemented a kind of Scripting Gauge called the Interoperability Gauge. This gauge uses the graph distance metric to rank the services based on their data compatibility (from output data schema to the input data schema of the connecting service). This gauge favors more specialized services over more generic services. For example, the output clusters of the Self-Organizing MAP (SOM) clustering service can be displayed by a generic hierarchical category visualization tool. However, there is a 2D SOM map visualization tool designed especially to visualize the clusters. The Interoperability Gauge ranks

the map visualization tool higher than the category visualization tool.

We have developed several ways to visualize the interoperability gauge results, as shown in Figures 5 (a) and 5 (b). We have augmented our service selection panel (bottom left portion of the Service Scripting Tool panel in Figure 5 (a)) by displaying a dial gauge next to each service choice summarizing its interoperability. The service schema panel (bottom left) is augmented to display the interoperability level of each input parameter of a particular service choice. Also, we have developed a ranked list panel that displays the service choices in order of interoperability, and a pop-up compound gauge (Figure 5 (b)) that details the individual input parameters with their subsumption relationships.

5.2 Compatibility Gauge

The SIM TBASSCO scripts are design to be portable — scripts generated in one computing service environment can be shared and executed in another environment. However, one problem that arises is that services available in the environment where the script is created may not be available in the execution environment. The Compatibility Gauge helps by finding potential replacement services. Given a particular service in a script, the Compatibility Gauge suggests replacement services based on the semantic closeness of their functionality, input and output.

A replacement service, *R*, is *strictly semantically compatible* with a service, *S*, if both services have the same functionality AND service *R* can accept any input that service *S* can accept, AND service *R* generates only outputs that service *S* can generate. For any script, a service may be substituted by a strictly semantically

compatible service without semantically affecting that script.

A strictly semantically compatible service is guaranteed to work across all possible scripts, however this compatibility requirement is overly restrictive if the script is known. *Context-dependent semantically compatible* service relaxes this definition by only requiring the replacement service, R, to accept any input that the predecessor services in the script can generate, AND to only generate output that the successor services in the script can accept. In terms of input this means that the replacement service does not have to accept all the input the original service can accept, as long as we can be sure that in the context of this script that such inputs will never be generated. In terms of output this means that the replacement service can generate outputs the original service does not, as long as we can be sure that the successor services in the script can handle the output.

The compatibility gauge is always used in the context of a script, so we adopt the less restrictive context-dependent compatibility. For example Figure 6 shows Compatibility Gauges that are applied to a document classification service, UAZ_NounPhraserExtractor to find out substitutable components of it. Figure 6 (a) is a sorted list of context-dependent semantically compatible services of the classification service. The top ranked compatible services include other phrase-based classification services, such as the place name extractor, keyword extractor and company name extractor. Figure 6 (b) is a compound gauge that shows detail compatibility levels (functional and I/O compatibility) of PlaceNameBasedClassification. However, in the strict semantic compatibility sense the UAZ_NounPhraserExtractor is unique. It is the only service that can be connected to UAZ's SOM clusterer service. If we had used strict semantic compatibility or if there were a SOM clusterer service attached to the UAZ_NounPhraserExtractor, then there would be no alternative compatible service available.

5.3 Insertion Gauge

GeoWorlds supports a library of component services in a framework intended to help users perform complex information analyses by identifying and applying available component services. Adding additional components to the system is complex, because of the number of existing components and the need for the new components to be interoperable. *Component developers* need help ensuring that offerings conform to appropriate service interfaces, and obey input/output data interchange syntax and semantics.

The Service Insertion Gauge indicates how well new components semantically integrate with existing components in the system, and what other components

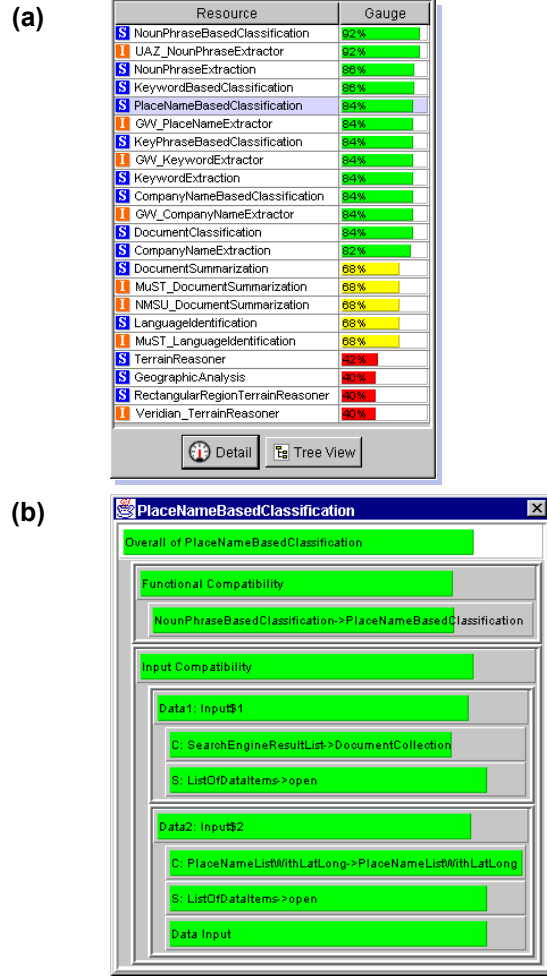


Figure 6: Compatibility Gauges: (a) Sorted list of compatible components; (b) Compound Gauge

may be needed to integrate the system more closely. The initial implementation uses the Interoperability Gauge to measure how many existing services understand the output of the new service, and how many existing services are able to generate output that the new services can accept.

GeoWorlds provides us with a testbed for evaluating of the utility and effectiveness of these tools. Recently, two new components, developed elsewhere, have been added to the GeoWorlds system. This exercise has provided us with an opportunity to test our Insertion Gauge.

BBN's Abstract Query Engine [2] provides a new service that is similar to query engines already present in GeoWorlds. As such, we would predict that it should score high on interoperability, since provisions for interfacing with other query engines have already been implemented. Application of the Insertion Gauge (Figure 7 (a)) yielded a high score, indicating that insertion would be relatively easy.

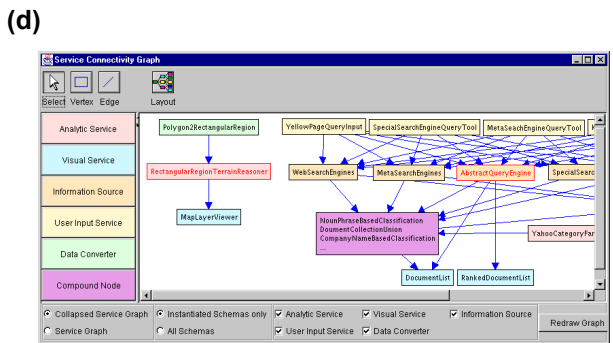
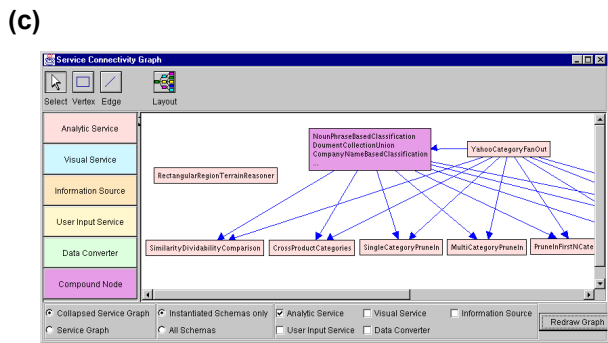
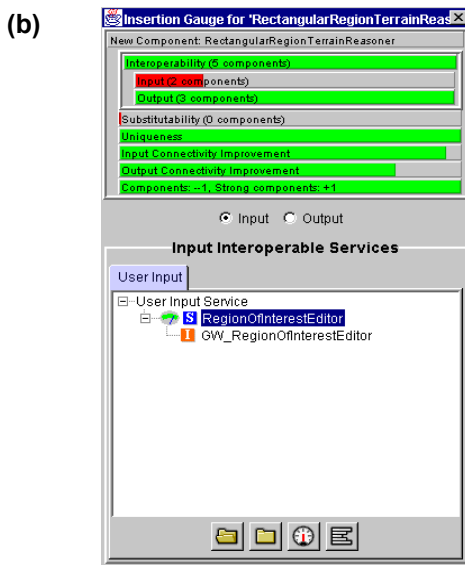
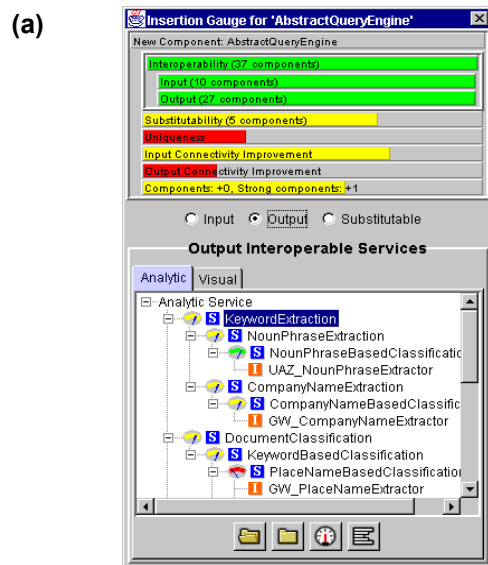


Figure 7: Insertion Gauges: (a) Compound Insertion Gauge Applied to BBN's Abstract Query Engine; (b) Compound Insertion Gauge Applied to Veridian's Terrain Reasoner; (c) Connectivity Graph for Veridian's Terrain Reasoner before adding a converter and visual service; (d) Connectivity Graph for Veridian's Terrain Reasoner after adding the converter and visual service

Veridian's Terrain Reasoner [24] offers a service that does not resemble any preexisting GeoWorlds components. It computes the difficulty of traversing land on foot. Figure 7 (c) shows the service connectivity graph of the Terrain Reasoner when it first introduced into GeoWorlds. It is an isolated node without any other components connected to it. Additional converters, input services and viewers are needed to properly invoke it. Figure 7(d) shows the service connectivity graph after additional interoperable services are added. However, even with the additional services the Terrain Reasoner still has low degree of service connectivity, and it is still separated from most of the services. The Insertion Gauge gives a lower overall score to the Terrain Reasoner, indicating that the effort to integrate this component into GeoWorlds would be relatively higher, and that interoperation with other components would be less. On the other hand, the fact that the Terrain Reasoner adds a service that did not exist previously is reflected in the creation of its own category within the subsumption

hierarchy; this yields a high score for the "uniqueness" dimension (Figure 7 (b)).

6. RELATED WORK

The knowledge-based approach that we propose has antecedents in a long line of research by others and ourselves. Neches has been an early advocate for modeling domain concepts to facilitate software reuse, interoperability and composition [19, 17]. Also, AI researchers were looking at generic descriptions of problem solving tasks to support composition and reuse of problem solving methods [4]. Marty Tenenbaum and his colleagues used ontologies to create customizable frameworks for instantiating manufacturing facility control systems [22, 8] Mark Musen used similar techniques for generation of medical information systems software [16]. We were early implementers of the concept, performing work on assembly of multiple logistics applications from a common component library

[10], user interface software development environments [18], and electronic commerce systems [23].

The last decade has witnessed an explosion of research in architectural description languages (ADL). Acme [9] seems to have emerged as the language for interchange among specific ADLs. We plan to piggyback on Acme to develop our semantic and performance extensions. All of the ADLs are capable of configuration view modeling of components and connector. Some also support execution view modeling. Rapide [14] is capable of generating a partial ordering set of events. MetaH [3] can perform task scheduling in real time guidance, navigation and control.

Web Services Description Language (WSDL) [5] is a proposal submitted to the W3C for using XML to describe network services as collections of communication endpoint (*ports*) capable of exchanging *messages*. Many elements of our approach and WSDL are comparable. The concept of a WSDL *operation*, consisting of an input message and an output message, corresponds to our concept of a service with *inputData* and *outputData* collection sets (see Figure 1). The *types* used to define the WSDL messages corresponds our document collection semantics, which we further refine to content and organization structure semantics. Although the WSDL proposal prefers the XSD (XML Schema) type system, it allows room for other type systems. We see our semantic reasoning mechanisms as a potential alternative typing mechanism that directly supports document collections. WSDL allows services to explicitly state *bindings*, such as SOAP, HTTP or MIME, to attach specific protocols and data formats to the messages, operations and endpoints. Currently, in our system we implicitly bind to JAVA using Jini/JavaSpaces entries. One area of our work that is missing from WSDL is our conception of templates that provides a composition mechanism to combine individual services to form higher-level services (see Section 4).

Major issues on the semantic interoperability in large object systems are summarized and discussed in [11]. Their work suggests that explicit representation and run-time manipulations of semantic information will reduce the time and effort to build a large software system that is composed of COTS and legacy components. Our metadata representation scheme, semantically based design gauges, and script-based modeling tools will provide such mechanisms to automate the semantic interoperability measurements and make possible to create system scripts and instances rapidly and efficiently.

The approach of using explicit semantic information for integrating heterogeneous software components has been used by various projects such as SIMS [1], InfoSleuth™ [21], and GINF [15]. They characterize the information sources by extracting the semantic information (domain ontologies and relationships between

components) and expressing it using high-level representation languages. When a user submit a query, their information mediators extract some semantic information from the query and analyze it for mapping the user request to appropriate information sources. Even though they provide mechanisms to perform meaningful information analysis functions for the data retrieved from the information sources, those functions are mostly information integration functions that are tightly bound with particular information sources. Our contribution lies in representation and analysis mechanisms for functional semantics of general software components.

7. CONCLUSION

This paper has described the initial version of a set of tools that support metadata-level modeling, a scripting mechanism, and a set of software gauges. These have been implemented in Java and integrated into the GeoWorlds system. GeoWorlds provides us with a testbed for evaluating the utility and effectiveness of these tools. The intent of these tools is to provide help for application developers to build programs (scripts) out of components, for component creators to register their products with component libraries and determine how those products fit in, and for system administrators to find ways to adapt software to limitations of their local environment.

At this point, the initial efforts have produced some compelling demonstrations of capability that suggest that the approach has promise. We have shown that a large software system used in an applied setting can be modeled, gauged and extended using the SIM TBASSCO tool set.

Detailed evaluation of these tools is an important piece of future work. We are interested in looking at the time required for design and redesign of customized information analysis services, for adding new components, and for switching among architectures. We are also interested in improvements to run-time performance facilitated by these tools, and the speed with which such changes can be effected. Plans are in progress for such an evaluation in the context of intended refinements to the GeoWorlds system, which has been our testbed for practical software development efforts.

8. REFERENCES

- [1] Y. Arens, C.A. Knoblock, and C.N. Hsu. Query Processing in the SIMS Information Mediator, Advanced Planning Technology, editor, Austin Tate, AAAI Press, Menlo Park, CA, 1996.
- [2] Abstract Query Engine Resource Page. BBN Technologies. <http://aai.bbn.com/ae.html>
- [3] P. Binns, M. Engelhart, M. Jackson, and S. Vestal. Domain-specific Software Architectures for Guidance,

- Navigation, and Control. *International Journal of Software Engineering and Knowledge Engineering*, vol 6 no 2, 1996.
- [4] B. Chandrasekaran. *Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design*. *IEEE Expert*, 1(3): 23-30, 1986.
- [5] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1.*, March 2001. <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- [6] M. Coutinho, R. Neches, A. Bugacov, K.T. Yao, V. Kumar, I.Y. Ko, R. Eleish, and S. Abhinkar. *GeoWorlds: A Geographically Based Information System for Situation Understanding and Management*. In *Proceedings of the First International Workshop on TeleGeoProcessing (TeleGeo '99)*, Lyon, France, May 1999.
- [7] M. Coutinho, R. Neches, A. Bugacov, K.T. Yao, V. Kumar, I.Y. Ko, R. Eleish, and S. Abhinkar. *GeoWorlds: A Geographically Based Situation Understanding and Information Management System for Disaster Relief Operations*. In *Proceedings of the 5th International Conference on Information Systems, Analysis and Synthesis (ISAS '99)*, Orlando, Florida, July 1999.
- [8] M.R. Cutkosky, J.M. Tenenbaum. *A Methodology and Computational Framework for Concurrent Product and Process Design*. *Mechanism and Machine Theory*, 25(3), 365-381, 1990.
- [9] D. Garlan, R.T. Monroe, and D. Wile. *Acme: An Architecture Description Interchange Language*. *Proceedings of CASCON '97*, November 1997.
- [10] B. Harp, P. Aberg, D. Benjamin, R. Neches, P. Szekely. *DRAMA: An Application of a Logistics Shell*. In *Proceedings of the Annual Conference on AI and Military Logistics*. Williamsburg, VA, 146-151, 1991.
- [11] S. Heiler, R.J. Miller, and V. Ventrone. *Using Metadata to Address Problems of Semantic Interoperability in Large Object Systems*. *First IEEE Metadata Conference*, Silver Spring, Maryland, April, 1996.
- [12] I.Y. Ko, R. Neches, and K.-T. Yao. *Semantically-Based Active Document Collection Templates for Web Information Management Systems*. In *Proceedings of the International Workshop on the Semantic Web*, September 2000, Lisbon, Portugal.
- [13] V. Kumar, A. Bugacov, M. Coutinho, and R. Neches. *Integrating Geographic Information Systems, Spatial Digital Libraries and Information Spaces for Conducting Humanitarian Assistance and Disaster Relief Operations in Urban Environments*. In *Proceedings of the Symposium on Advances in Geographic Information Systems (ACM-GIS'99)*, Eighth International Conference on Information and Knowledge Management (CIKM'99), Kansas City, Missouri, November 1999.
- [14] D.C. Luckham, J.J. Kenney, L. M. Augustin, J. Vera, D. Bryan, W. Mann. *Specification and Analysis of System Architecture Using Rapide*. *IEEE Transaction on Software Engineering*, Vol 21, No 4, April 1995.
- [15] S. Melnik et al., *Generic Interoperability Framework*, Working Paper, Department of Computer Science, Stanford University.
- [16] M.A. Musen. *Overcoming the limitations of role-limiting methods*. *Knowledge Acquisition*, 4(2):165--170, 1992.
- [17] R. Neches, R. Fikes, T. Finin, R. Patil, T. Senator, and W.R. Swartout. *Enabling Technology for Knowledge Sharing*. *AI Magazine*, 12(3), 1991, 36-56.
- [18] R. Neches, J.D. Foley, P. Szekely, P. Sukaviriya, P. Luo., S. Kovacevic, and S. Hudson. *Knowledgeable Development Environments Using Shared Design Models*. In *Proceedings of the ACM/AAAI International Workshop on Intelligent User Interfaces*, Orlando, FL, January, 1993.
- [19] R. Neches. *Knowledge Sharing in Integrated User Support Environments: Applications, Frameworks, and Infrastructure*. In K. Fuchi and T. Yokoi (Eds.), *Proceedings of the 1993 International Conference on Building and Sharing of Very Large-Scale Knowledge Bases*, Tokyo, Japan: Japan Information Processing Development Center (JIPDEC), December 1-3, 1993.
- [20] R. Neches, S. Abhinkar, F. Hu, R. Eleish, I.Y. Ko, K.-T. Yao, Q. Zhu, and P. Will. *Collaborative Information Space Analysis Tools*. *D-Lib Magazine*, October 1998, ISSN 1082-9873.
- [21] M. Nodine, W. Bohrer, and A. H. Hiong Ngu. *Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth™*. *15th International Conference on Data Engineering*, March, 1999, Sydney, Australia.
- [22] J.Y.-C. Pan, J.M. Tenenbaum, and J. Glicksman. *A Framework for Knowledge-Based Computer-Integrated Manufacturing*. *IEEE Transactions on Semiconductor Manufacturing*, 2(2), 33-46, 1989
- [23] C. Powley, D. Benjamin, D. Grossman, R. Neches, P. Postel, E. Brodersohn, R. Fadia, Q. Zhu, and P. Will. *DASHER: A Prototype for Federated E-Commerce Services*. *IEEE Internet Computing*, Vol 1, No 6, November/December 1997.
- [24] *Tactical Terrain Analyzer*. Veridian. http://www.veridian.com/products/tactical_terrain_analyzer.asp
- [25] K.T. Yao, R. Neches, I.Y. Ko, R. Eleish, and S. Abhinkar. *Synchronous and Asynchronous Collaborative Information Space Analysis Tools*. In *Proceedings of the International Workshop on Collaboration and Mobile Computing (CMC'99)*, September 1999, University of Aizu, Fukushima, Japan.
- [26] K.T. Yao, I.Y. Ko, R. Eleish, and R. Neches. *Asynchronous Information Space Analysis Architecture Using Content and Structure Based Service Brokering*. In *Proceedings of Fifth ACM Conference on Digital Libraries (DL 2000)*, San Antonio, Texas, June 2000.