# Composition as a Service

**M. Brian Blake** • *University of Notre Dame*
**Wei Tan** • *University of Chicago*
**Florian Rosenberg** • *CSIRO ICT Centre, Australia*

Understanding reusable software and assets within and among multiple organizations' system infrastructures can be challenging. IT systems might be widely distributed, and interconnections can be complicated. Ever-evolving computing technologies reduce an organization's in-house expertise for reusing software, even when leveraging software systems within its own firewalls. As such, the on-demand integration of software and capabilities might benefit from an outsourcing paradigm — described here as composition as a service.

Services computing[1] is an emerging paradigm in which services that humans typically execute are realized as network-accessible software applications or Web services. Web service composition is the ability to integrate multiple services into higher-level applications. This technology has seen increased attention in published works, and researchers have introduced numerous approaches to automate it.[2] As such, current techniques suggest that organizations will acquire related tools and perform integration activities locally. The knowledge and expertise to perform composition activities is hard to attain and equally difficult to maintain. Given services' proliferation as well as their format and usage diversity, traditional approaches seem rigid and not easily adoptable; such approaches assume that all the information required to conduct service composition, such as service portfolios, data schemas, or ontologies, are available locally to a given user. Thus, the knowledge obtained during service composition isn't stored, disseminated, or shared. Because service composition is a knowledge-intensive task, leveraging the services metadata, I/O data schema, and, particularly, experience embedded in already-built workflows can significantly improve both performance and user experience. To be competitive, organizations must be able to transfer and reuse knowledge attained after each composition scenario.

Current challenges in automating service composition neglect another important paradigm in which a third-party entity can provide a service on a service provider's behalf — *software as a service* (SaaS).[3] Centralizing composition activities into a first-class service alleviates the previously mentioned challenges. SaaS would provide a solution for two scenarios, in particular. One occurs when an organization is searching for a particular software capability to interact within an established business process. The organization might submit an incomplete workflow specification and request a capability that can complete the process. In the second scenario, multiple organizations might have only written specifications. Two organizations could submit relevant application specifications and request potential overlap in their service offerings. The former scenario suggests the use of a third-party organization to integrate software, whereas the latter requires text manipulation and matching techniques to search for common specifications. Both scenarios require an SaaS customization entitled *composition as a service* (CaaS).

## A Protocol for Reusing Historical Composition Experiences

Florian Rosenberg and his colleagues first introduced the idea of CaaS[4] in a paper in which they present a composition service for avoiding the need to install a client-side composition infrastructure. The framework focuses

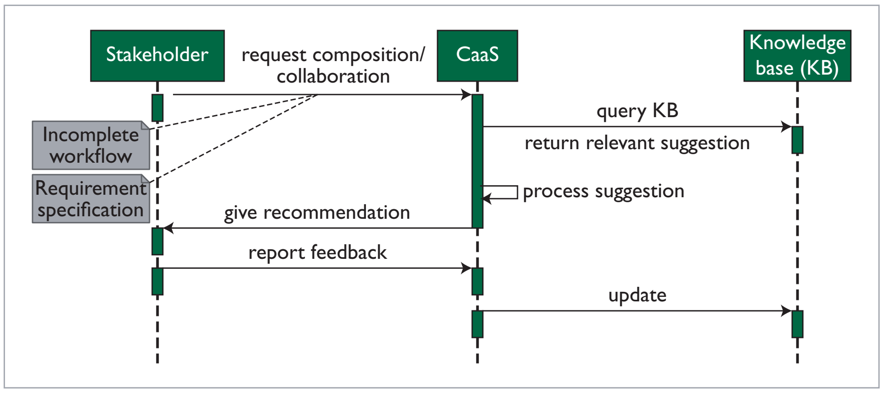

*Figure 1. Abstract interaction when accessing historic composition information. Although this figure suggests one stakeholder, composition as a service (CaaS) can operate explicitly on the input multiple stakeholders provide.*

mainly on quality-of-service (QoS)-aware composition using a domain-specific language. Here, we extend the scenario to a broader scope to cover both service and capability composition. Considering a stakeholder who needs to find and consume a required capability within its own offerings, the input to CaaS might be an incomplete business process or a general description of those offerings. A CaaS capability could acquire the previously mentioned artifacts as input and suggest a specific service workflow (in terms of software) or related application specifications (in terms of written specifications). Stakeholders can report the effectiveness of the recommendations to the CaaS capability; organizations can maintain a knowledge base (KB) to facilitate reusing previously used software, specifications, and composition routines in the future. Figure 1 shows this interaction. Although the figure suggests one stakeholder, CaaS can operate explicitly on the input that multiple stakeholders provide. Furthermore, the CaaS approach can learn from historical composition information to augment future recommendations.

## CaaS for Software Composition

Current approaches to service composition are generally used in isolated, standalone applications. Knowledge can neither accumulate nor be shared among people who might undertake similar composition tasks. Web services are available in the business domain — for example, Google (http://code.google.com/apis/ajax/) and Amazon (http://aws.amazon.com) Web services — as well as scientific domains, such as Web services from the European Bioinformatics Institute (www.ebi.ac.uk/Tools/webservices/), the US National Center for Biotechnology Information (NCBI; www.ncbi.nlm.nih.gov/entrez/query/static/esoap_help.html), the Cancer Bioinformatics Grid (caBIG) sponsored by the US National Cancer Institute (see http://cagrid-portal.nci.nih.gov), and bioCatalog (www.biocatalogue.org), which maintains a comprehensive catalog of biological Web services. CaaS could be instrumental in recommending where services can be integrated across the previously mentioned domains.

In short, CaaS is a service that mediates communication between multiple clients. It provides composition recommendations to stakeholders and collects feedback from them. At the client side, users will have to obtain a CaaS plug-in to the modeling tool of their integrated development environment (IDE). This plug-in acts on the user's behalf to detect his or her requirement and forward it in an abstract and client-independent format to the CaaS service. For example, if a user adds a `<receive>`, a `<reply>`, or an `<invoke>` activity in a Business Process Execution Language (BPEL) process, this could provide an input query to the CaaS service. When CaaS returns a list of recommendations, the plug-in translates it back to a format the user or client can comprehend. These suggestions might include a completed workflow, additive fragments, or suggestions on modifying current workflow. Users can choose among multiple recommendations and optionally send the selection with other feedback to CaaS.

The CaaS service itself relies on a KB to aggregate knowledge and provide suggestions. The KB manages four categories of information:

- *Basic service portfolio.* The service registry, indexing services, and individual services end points (that is, Web Service Description Language [WSDL] files) contain signatures of services and basic information on how to use them.
- *Enhanced service data.* Some services have additional metadata on them, such as semantic annotation on data types or service usage information given in OWL-S (www.daml.org/owl-s/).
- *Workflow repository.* Certain application domains — for example, an enterprise or a bioinformatics research group — might have repositories of workflows already built and used. These are good reference cases for others who deal with similar business or scientific experiments.
- *User portfolio and feedback.* A user portfolio contains user information (identity, institution, and so on) and feedback on using CaaS, which helps CaaS provide more personalized suggestions. For example, given user feedback

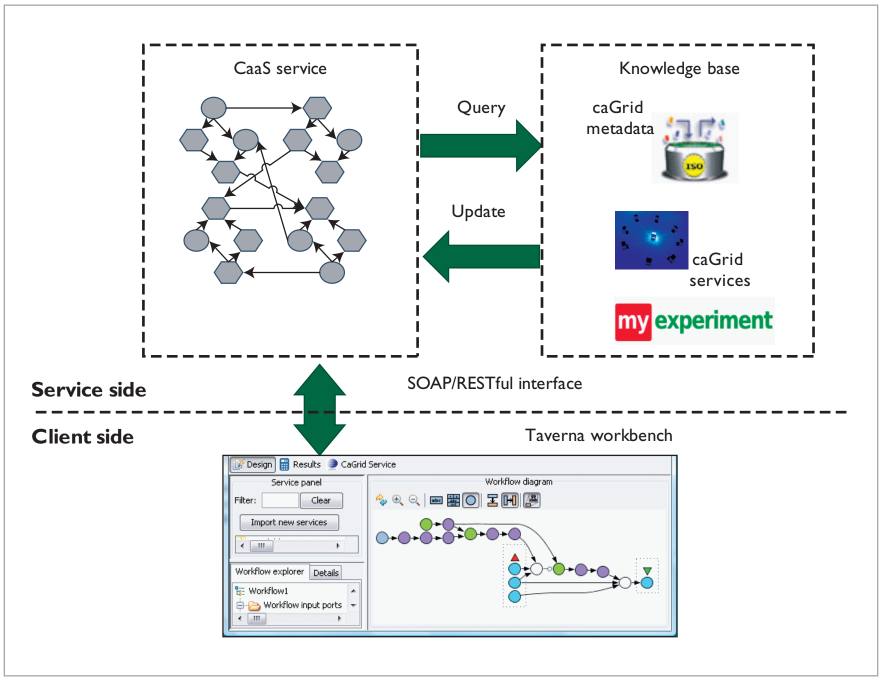

*Figure 2. Implementation of composition as a service (CaaS) in caGrid. At the client side, users model workflows in the Taverna workbench, including the part that they aren't quite clear about and ultimately require assistance with. A Taverna plug-in inside the workbench communicates with the CaaS service, issuing assistance requests, retrieving suggestions, and giving feedback, using SOAP or Representational State Transfer (RESTful) interfaces.*

on a selection from a list of Blast services, CaaS might conclude that users from the University of Chicago prefer NCBI-hosted Blast services, and can use this preference setting on suggestions to users from the same group.

In the next section we'll discuss a real case in which the idea of CaaS is used to facilitate the composition of scientific workflows in biomedical research.

## A Case Study from caGrid

Currently, we're working on implementing CaaS in the context of the caGrid (which is the aforementioned caBIG's grid infrastructure) workflow system (see Figure 2).[5] The caGrid workflow system was developed based on the Taverna workbench (http://taverna.sourceforge.net) to orchestrate caBIG grid services.

The application framework is divided into a client and a service side. At the client side, caGrid users model workflows in the Taverna workbench. They employ abstract data and service elements to model the uncertain part of the workflow that they aren't quite clear about and ultimately require assistance with. Besides those abstract elements, a workflow also contains concrete services and data that users already know and want. A Taverna plug-in inside the workbench communicates with the CaaS service while issuing assistance requests, retrieving suggestions, and giving feedback, using SOAP or Representational State Transfer (RESTful) interfaces.

At the service side, the CaaS service's core functional part is called a *service net*, which acts as an index of all the metadata, services, and workflows. The nodes in a service net are operations or data elements, and the edges between them are classified into three categories:

- *data-data edges* represent data relations defined in the WSDL schema and caGrid metadata;
- *operation-data edges* represent service operations' I/O data; and
- *operation-operation edges* represent connections between operations in existing workflows.

The KB contains a collection of caGrid services, caGrid metadata (a shared vocabulary and its annotations on caGrid services), and a caGrid workflow repository hosted at the myExperiment (www.myexperiment.org) Web site.

The CaaS service inspects the abstract workflow users submit from the workbench and finds similar graphs, or *skeletons*, in the service net. It then uses a skeleton to query the KB to get more data (such as a data node schema or operation node URL), combines them with skeletons to produce meaningful recommendations, and then returns them to the client. When any update occurs in the KB, the service net must be updated accordingly.

## CaaS for Written Specification Aggregation

Another implementation of the CaaS architecture is the ability to search for correlations in multiple organizations' applications and developmental specifications (that is, written specifications). Imagine that multiple organizations exist within the same domain, initiating new software development projects. An initial step in these projects is to elicit requirements for the new applications. Within an organization or across multiple organizations, overlap across these requirements documents is difficult to ascertain. Organizations can use CaaS to search for similar requirements across multiple requirements documents. This approach is beneficial in that architects and developers can identify redundancies in software development activities early in the life cycle. Although the scenario and supporting application here identify requirements documents, CaaS can leverage any text-based description in this approach. Figure 3 illustrates an application that can correlate software requirements specification documents in search of shared capabilities. The Venn diagram represents organizations' boundaries. The overlap between the circles demonstrates where significant overlap exists between multiple organizations and indicates a potential for collaboration or software reuse. The smaller dots represent capabilities inferred from words and phrases used in the requirements documents. Dots in the overlap of the Venn diagram represent potential shared capabilities.

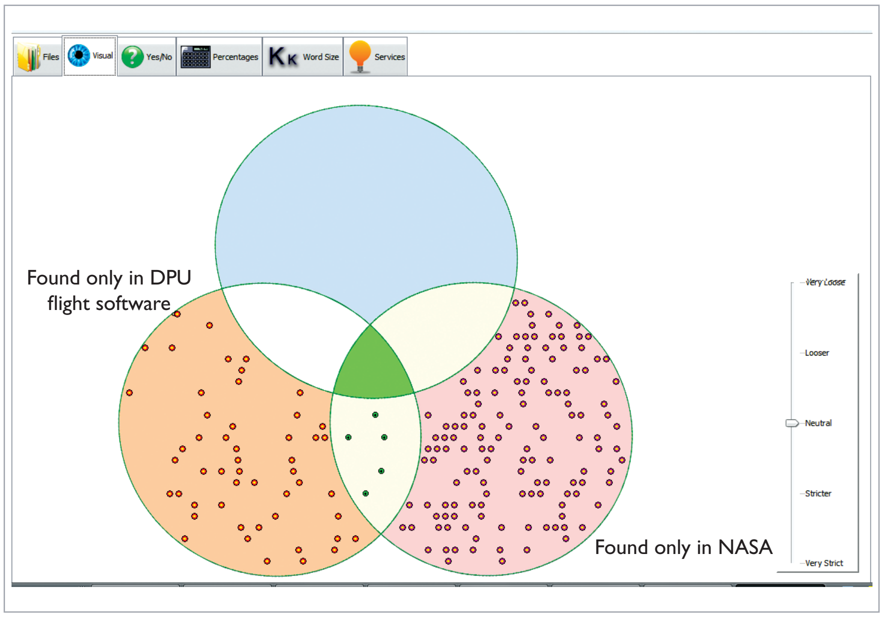

*Figure 3. A composition-as-a-service (CaaS) requirements aggregation approach. The tool in this screenshot uses a Venn diagram to show similar requirements between two different organizations. Requirements at the intersection of the diagram represent requirements predicted to be shared by the two organizations.*

## CaaS Research Challenges

Increased adoption of the SaaS paradigm and the proliferation of available Web services require new paradigms that rapidly compose new software systems from existing artifacts. The proposed CaaS approach attempts to combine collaborative software engineering principles with recent advances in service-oriented computing. Thus, it poses a variety of research challenges.

### Shared Artifact Repository

The first challenge is the provisioning of a lightweight approach to build an extensible and shared artifact repository — for example, consisting of a KB for managing common vocabulary, workflow templates, and service descriptions. Additionally, relationships between these artifacts, user information, and their requirements must be managed and automatically inferred from existing knowledge. Low-cost and rapid accessibility in a Web 2.0 manner should enable easier adoption and integration into existing tools (such as IDEs).

### Context-Awareness

In CaaS, the importance of context is manifold — including users' incomplete workflows and workflow requirements as well as their identities, service preferences, and usage information. Furthermore, CaaS can leverage social network information to gather additional context about users' behavior. It's crucial to provide mechanisms that infer how CaaS has obtained context information to make its service composition approach more intelligent and sensitive to users' requirements.

### Feedback, Recommendation, and Trust

Feedback mechanisms are important means for incorporating service and workflow ratings into the KB. This lets the CaaS approach devise service or workflow recommendations for users when combined with their context information. An important aspect for feedback and recommendation is trust to ensure that feedback can't be forged or incorrect such that CaaS provides false recommendations.

### Visualization

Different ways to visualize and enhance the user experience when using CaaS can streamline requirements elicitation among different organizations or illustrate common or redundant information (such as services or incomplete workflows). Intelligent methods for visualizing these artifacts can help reduce the overall workflow development time and increase developer satisfaction.

Current outsourced software engineering revolves around rapid application development and software maintenance activities. CaaS suggests that the next evolution will let organizations manage systems integration in much the same vein.

**Acknowledgments**

M. Brian Blake thanks Alex Yale-Loehr and Ian Schlesinger for their development activities with respect to the requirements tool representing the composition-as-a-service (CaaS) paradigm. Wei Tan thanks Ian Foster, Ravi Madduri, and Dinanath Sulakhe for the constructive discussions on CaaS in caGrid. Florian Rosenberg thanks Anton Michlmayr, Philipp Leitner, and Predrag Celikovic for their fruitful discussions on CaaS and its implementation for quality-of-service-aware service composition.

## References

1. L.-J. Zhang, J. Zhang, and H. Cai, *Services Computing*, Springer, 2007.
2. W. Schreiner and S. Dustdar, "A Survey on Web Services Composition," *Int'l J. Web and Grid Services*, vol. 1, no. 1, 2005, pp. 1–30.
3. M.P. Papazoglou and G. Georgakapoulos, "Service-Oriented Computing," *Comm. ACM*, vol. 46, no. 10, 2003, pp. 24–28.
4. F. Rosenberg et al., "Towards Composition as a Service – A Quality-of-Service-Driven Approach," *Proc. 25th Int'l Conf. Data Eng.* (ICDE 2009), IEEE CS Press, 2009, pp. 1733–1740.
5. W. Tan, I. Foster, and R. Madduri, "Combining the Power of Taverna and caGrid: Scientific Workflows that Enable Web-Scale Collaboration," *IEEE Internet Computing*, vol. 12, no. 6, 2008, pp. 61–68.

**M. Brian Blake** is a professor in the Department of Computer Science and Engineering, College of Engineering, at the University of Notre Dame. He conducts applied research in the areas of service-oriented computing and enterprise integration in collaboration with various government and industry organizations. Blake has a PhD in information and software engineering from George Mason University. He's a senior member of the IEEE. Contact him at m.brian.blake@nd.edu.

**Wei Tan** is a research staff member in the Computation Institute at the University of Chicago and Argonne National Laboratory. His research interests include scientific workflow, service-oriented computing, and Petri nets. He's now involved in multiple health-informatics-related projects, providing scientific workflow solutions for domain users. Tan has a PhD in automation engineering from Tsinghua University, China. Contact him at wtan@mcs.anl.gov.

**Florian Rosenberg** is a research scientist at the CSIRO ICT Centre in Australia. His general research interests include service-oriented computing and software engineering, particularly all aspects related to quality-of-service-aware service composition and adaptation. Rosenberg has a PhD in computer science from the Vienna University of Technology. Contact him at florian.rosenberg@csiro.au.