# Blackboard-style Service Composition with Onto⇔SOA

Maksym Korotkiy[1] and Jan Top[1,2]

[1] Vrije Universiteit Amsterdam, Department of Computer Science
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
maksym@cs.vu.nl jltop@few.vu.nl
[2] Wageningen UR - A&F, P.O. Box 17, 6700 AA Wageningen, The Netherlands
jan.top@wur.nl

**Abstract.** We propose an approach to service composition based on the ideas from Blackboard Systems extensively investigated in AI in 1970-80s. We combine these ideas with Onto⇔SOA – a SOA *design* framework that relies on a restricted document-oriented and ontology-based service model to provide guidelines for construction of services. The proposed Blackboard-style composition approach requires neither extensive service model nor explicit workflow specification and enables composite functionality to emerge by bringing a number of services together and making them interact via a shared data repository. We illustrate that a Blackboard-style mechanism combined with a restricted service model is a feasible approach for non-trivial service composition scenarios. In such a scenario, described in the paper, we compose a number of services to check consistency of units of measurement in mathematical statements.

## 1 Introduction

Services are seen as highly reusable software components that can be effectively composed providing complex functionality to their consumers. There exist a variety of SOA frameworks and approaches to service composition. The field of Semantic Web Services (SWS) is closest to the area of our research. OWL-S [1] and WSMO [2] are two most well-known SWS approaches. Both of them rely on extensive ontology-based semantic service models to *automate* tasks such as discovery, invocation, choreography and orchestration of Web Services. The extensive formal frameworks defined by these approaches achieve, to a certain extent, the goal of automated service composition [3]. However, as these frameworks aim to cover the widest possible range of services, they tend to become highly complex which hinders their overall acceptance.

In [4] we have proposed Onto⇔SOA – a SOA design framework focused on general SOA characteristics such as *domain alignment* and *loose coupling*, which are considered crucial for delivering re-usable functional components. Like SWS, Onto⇔SOA combines ontologies and SOA. However, unlike SWS, it defines a *restricted* service model and employs an ontology *directly* as the conceptual description of the service interface. In Onto⇔SOA services communicate in terms of documents containing relevant domain concepts only. In Onto⇔SOA we maintain a simple and consistent approach to service design, such that can be naturally integrated with ontologies.

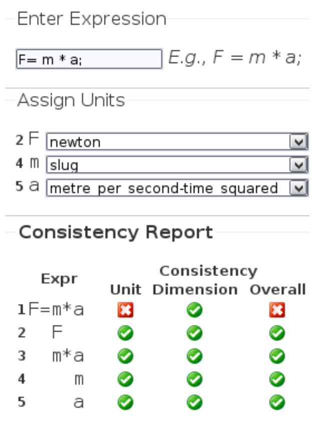
So far we have not explicitly addressed the issue of service composition in the Onto⇔SOA framework. In this paper we fill this gap by further extending the framework with a composition mechanism based on Blackboard Systems. Blackboard Systems are best explained by an analogy with a team of experts cooperatively solving a complex problem on a blackboard. The experts are independent, belong to different domains and do not directly interact with each other. Instead, they observe the current state of the problem solving process captured on the blackboard, and contribute to it by applying their domain knowledge. Blackboard Systems exhibit the ability to supply a general and flexible composition mechanism capable of organizing multiple components in a variety of application areas (speech recognition, process control, case-based reasoning etc). Hence, we have decided to investigate if (and how well) Blackboard-based composition works with Onto⇔SOA services.

More specifically, in this paper we demonstrate that:

- The Blackboard-based mechanism is a viable approach to service composition in SOA.
- Despite being intentionally restrictive (e.g., preconditions and effects of a service are not described) the service model in Onto⇔SOA is suitable for service composition. More generally, we submit that in many non-trivial scenarios no extensive description of a service model or workflows is required to enable effective composition of services.

To support these claims we employ a use case from the e-Science domain. The use case addresses the problem of detecting inconsistent use of units of measurement in mathematical statements. To solve this task we implement a number of Onto⇔SOA services and combine them using a Blackboard-based approach.

Thus, we organize the paper as follows. In Sec. 2 we describe the use case. After that, in Sec. 3, we briefly outline a service model behind Onto⇔SOA. In Sec. 4 we describe the main components of traditional Blackboard Systems. Then, we adjust the traditional Blackboard composition mechanism to Onto⇔SOA in Sec. 5 and apply it to the use case in Sec. 6. We elaborate on the design of two services in Sections 6.1 and 6.2, and describe a sample composition run in Sec. 6.3. Finally, we discuss some issues of the proposed Blackboard-style service composition in Sec. 7 and conclude with Sec. 8.



Enter Expression

F= m * a; E.g., F = m * a;

Assign Units

| | | |
|---|---|---|
| 2 | F | newton |
| 4 | m | slug |
| 5 | a | metre per second-time squared |

Consistency Report

| | Expr | Consistency | | |
|---|---|---|---|---|
| | | Unit | Dimension | Overall |
| 1 | F=m*a | ✗ | ✓ | ✗ |
| 2 | F | ✓ | ✓ | ✓ |
| 3 | m*a | ✓ | ✓ | ✓ |
| 4 | m | ✓ | ✓ | ✓ |
| 5 | a | ✓ | ✓ | ✓ |

**Fig. 1.** A screenshot of the unit consistency checking demo application.

## 2 Use Case: Checking Consistency of Units of Measurement in Mathematical Statements

In many engineering and scientific applications consistent use of units of measurement is an important quality assurance tool. There are numerous examples of severe losses resulted from inconsistent use of units of measurement. To name one, we can refer to a 125$ million Mars orbiter lost in 1999 because engineering teams used units from different measurement systems [3]. A loss like this could have been prevented if an automated unit consistency checking were available.

To determine consistency [4] of an expression (e.g., $F = m \times a$) we have to know what units are assigned to each of the variables (e.g., $F$ – $Newton$, $m$ – $kilogram$, $a$ – $meter/second^2$). Given this information we can apply knowledge from the domain of units of measurement and determine consistency of that expression (e.g., $Newton$ can be expressed as $kilogram \times meter/second^2$, hence the assignment is consistent).

For this use case we design a demo application [5] that relies on a consistency checking service composed from a number of independent services. The implemented workflow consists of three steps which are clearly recognizable in the GUI depicted on Fig. 1: **1)** the user types in an expression, **2)** assigns units of measurement to variables employed in the expression, and finally **3)** starts the consistency checking procedure.

## 3 Onto⇔SOA in a Nutshell

We have proposed the Onto⇔SOA framework [4] as a simple yet effective ontology-enabled SOA style. We aim to improve reusability of services by enforcing their *domain alignment* and *loose coupling* characteristics. To achieve that we have defined a restricted service model that exposes only a conceptual service interface captured in a schema to which we refer as a *service ontology* (Fig. 2).

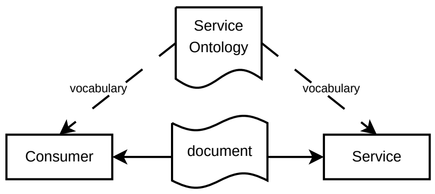


**Fig. 2.** Main components of the Onto⇔SOA framework.

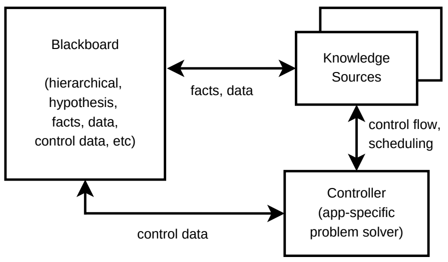

Blackboard
(hierarchical, hypothesis, facts, data, control data, etc)
facts, data
Knowledge Sources
control flow, scheduling
control data
Controller (app-specific problem solver)

**Fig. 3.** Major components of a traditional Blackboard System.

The service model in Onto⇔SOA requires a service to be *document oriented*, in contrast to communication in terms of procedure calls (RPC). A service ontology, therefore, is a specification of a document-oriented service interface. It describes vocabulary employed in documents communicated to/from a service and exposes no other details about a service such as preconditions, effects, process model etc.

[3] http://www.cnn.com/TECH/space/9909/30/mars.metric.02/

[4] Here and further on we use consistency to refer to consistency of units of measurement and dimension.

[5] The demo is accessible at http://swpc333.cs.vu.nl:8080/usdemo/ .

In Onto⇔SOA we explicitly require services to be *coarsely grained* and *independent* from each other. This can be achieved by limiting (ideally to one) a number of services designed for a discrete application domain.

We further establish a simple and application-independent *connector* mechanism by requiring a service to be *session stateless* and restricting it to a single operation (inference of new facts). Additionally, we limit the interaction between a service and its consumer to the request-response protocol.

We have operationalized Onto⇔SOA in *MoRe* [5], where a service ontology is expressed in RDFS [6] and RDF [7] documents are communicated between a consumer and a REST [8]-like service. We have applied *MoRe* to a number of use cases from the e-Science domain [5, 9]). So far, in Onto⇔SOA we have only addressed the task of service invocation. Service discovery and composition are not addressed in the core framework. However, we believe that these tasks will be also facilitated if we enforce the domain alignment and loose coupling characteristics of a service.

## 4 Blackboard Systems

Blackboard Systems have been extensively researched in AI in 1970-80s (see [10] for an extended introduction into the field). The main idea behind Blackboard Systems can be illustrated by imagining a team of experts that cooperatively solve a problem via a blackboard. The experts are allowed to interact via the blackboard only and their access to the blackboard is managed by a dedicated controller. Thus, we can distinguish three main elements in a Blackboard System: the Knowledge Source (expert), the Blackboard and the Controller (Fig. 3).

*Knowledge Sources* (KS) are mutually independent functional components capable of inspecting and modifying the Blackboard. KS represent experts from different domains that cooperate to solve a complex problem. In many Blackboard Systems a KS consists of trigger and action procedures. The trigger procedure allows a KS to determine if a blackboard contains facts sufficient to contribute to it. The purpose of a trigger is similar to the purpose of service preconditions employed in Semantic Web Services (SWS) approaches. Triggers enable the Controller to schedule KS to achieve the most efficient problem-solving process. However unlike preconditions in SWS, triggers are normally not specified declaratively and not employed for automatic construction of workflows.

The *Blackboard* is a heterogeneous repository shared by all KS and the Controller. This repository enables cooperation among KS, and serves as a temporary buffer. The Blackboard can contain symbolically represented and, often, hierarchically organized solution space as well as control data employed by the Controller. The structure of the Blackboard is usually application specific to achieve the most efficient communication among KS and the Controller. It is assumed (rather implicitly) that there is a certain syntactic and semantic compatibility between KS. This allows them to (at least partially) understand the content of the Blackboard and extend it with new facts, which in turn can be understood by other KS.

The *Controller* synchronizes and coordinates KS to establish an effective and efficient problem-solving process. Overall application-specific problem-solving strategy is normally embedded into the Controller. The strategy is flexible enough to enable an arbitrary scheduling of KS that is decided upon by the Controller on the basis of trigger procedures.

In AI the following benefits of Blackboard Systems are emphasized [11, 12]:

- Blackboard Systems are arguably considered to be the most general and flexible architecture for building knowledge-based systems.
- Blackboard Systems provide for an excellent integration framework for components (Knowledge Sources) that employ heterogeneous representations and expertise. This characteristic of Blackboard Systems is very attractive for Enterprise Application Integration for which SOA is also often employed.
- Separation of concerns between the Controller and KS allows a Blackboard System to make dynamic control decisions. This property is also very attractive for SOA deployed in the very dynamic environment of the Web.
- The inherent modularity of a Blackboard System and independence of KS provide for significant software-engineering benefits.

Blackboard Systems have been applied in numerous application areas such as process control, planning and scheduling, speech recognition etc. As summarized in [11] the main disadvantage of Blackboard Systems is that they do not scale down very well to simple problems. In addition, they are considered to be useful only during prototyping. For performance reasons, production systems are usually re-implemented with more conventional means providing for higher performance. Finally, the components of traditional Blackboard Systems appeared to be rarely re-used. In most cases Blackboard Systems were designed from scratch. Most of these perceived disadvantages reflect performance considerations which are of lesser concern in SOA where re-usability is the ultimate goal.

## 5 Blackboard-style Service Composition in Onto⇔SOA

The generality and flexibility of Blackboard Systems make the underlying composition mechanism very attractive to modern Service-Oriented Architectures. Although Onto⇔SOA services share many properties with Knowledge Sources (KS), there are also considerable differences between the organization of Blackboard Systems and SOA:

- In Blackboard Systems the KS are designed to work together on a specific, predefined task. On the other hand, SOA services are not aware of the complex application scenarios in which they will participate.
- Blackboard Systems emphasize flexible and dynamic control. Reusability of KS is not a design goal as such. Contrary to this, in SOA reusability of services is the most important concern.

Therefore, we will have to adapt the Blackboard-style composition mechanism to apply it in Onto⇔SOA. In the following subsections we describe the three main elements of Blackboard-style composition as applied in Onto⇔SOA: the Controller, services acting as KS and the Blackboard.

### 5.1 The Controller

We begin by defining the Controller because it requires considerable adjustments to meet the restrictions of the Onto⇔SOA service model. According to this model the Controller belongs to the connector mechanism, and hence must be designed as a *simple* and *application-independent* component. The Controller can be employed either as a stand-alone service or as part of another service that internally performs service composition. In both cases, however, the Controller itself does not contain any application-specific functionality. This functionality can be contained either in one of the composed services or in the service consumer.

The application-independent Controller allows Onto⇔SOA services to be easily combined across application contexts. We assume that the Controller knows which services participate in composition but not what they do. At this point we do not elaborate on how the Controller discovers the services to be composed.

In Onto⇔SOA the Controller uses the following basic composition procedure:

1. The Controller sequentially invokes all services to be composed in a *non-predefined order*. Each service is invoked at least one time. During each invocation the Controller sends the complete content of the blackboard to one of the services as an input document. The output document of the service is then moved to the blackboard.
2. Another invocation sequence follows, if the content of the blackboard has been modified after invoking all services. Otherwise, the process stops.

Presently, to maintain simplicity and to avoid concurrency related problems, we assume that during composition only one service may access the blackboard.

Since the Controller contains no application-specific logic and the composed services expose their conceptual interfaces only, the Controller cannot predict whether a service can contribute to the blackboard at a given iteration. Hence, unproductive invocations – service invocations that do not add new facts to a blackboard – cause additional overhead. The Controller can reduce this overhead by inspecting the changes on the blackboard and adjusting the service invocation order for next iterations. We discuss this issue in some detail in Sec. 7. However, finding a generally applicable composition optimization mechanism is outside the scope of our work at the moment.

### 5.2 The Onto⇔SOA Service as a Knowledge Source

Knowledge Sources are traditionally seen as domain-specific problem solvers. In the Onto⇔SOA framework we treat services in a more general way as experts capable of applying their knowledge to infer facts about the domain of discourse. A few additional assumptions and clarifications are required to enable the participation of Onto⇔SOA services in the Blackboard-style composition as described in the previous subsection.

First, the services must be compatible with each other on the conceptual (semantic) and data-model levels. In *MoRe*, a specific implementation of Onto⇔SOA, we use the RDF data-model to express documents. To achieve semantic compatibility between the collaborating services we require that each service ontology overlaps with at least one other service ontology. Both semantic and data-model incompatibility can be, in principle, resolved via mediation mechanisms which we, however, do not consider to be part of the framework.

Second, the introduced composition procedure terminates when none of the services can modify the blackboard anymore. We take measures to reduce chances that (conflicting) interaction between services prevents normal composition termination. For this we require that the services may only add new facts to the blackboard. They may neither remove nor modify existing facts. In addition, to prevent infinite expansion of the blackboard we require that a service does not modify an document generated by the same service: e.i. submitting an output document of a service to the same service will not modify the document.

However, there are still many scenarios when composition does not terminate normally. In those cases termination could be achieved by enforcing a maximum number of service invocations. Such kind of abnormal termination is not preferred because it would require us to expose internal details of a composition procedure hindering both domain alignment and loose coupling of such a service.

Finally, in Onto⇔SOA we favor session-stateless services. This implies that the collaborating services should not use blackboard as a buffer to store intermediate results.

### 5.3 The Blackboard

The Blackboard contains information (a collection of facts) shared between composed services. In Onto⇔SOA the Blackboard has a homogeneous structure, compatible with the data-model employed in the documents communicated to and from the collaborating services. For example, if we aim to compose *MoRe* services then the corresponding blackboard must have the RDF data model. The Blackboard is conceptually neutral – it does not enforce any conceptual structure (e.g., hierarchically organized solution space as in traditional Blackboard systems etc).

The Blackboard is used exclusively to enable interaction between services. As stated previously, we do not allow it to serve as a temporary buffer for intermediate results internal to the respective services. Nor may it contain control data.

In traditional Blackboard Systems the blackboard structure is application specific to achieve an efficient problem-solving process. Contrary to this, in Onto⇔SOA we emphasize generality and reusability (consequently sacrificing some efficiency), and, therefore require the blackboard structure to be application independent.

## 6 Use Case Implementation

We apply the proposed Blackboard-style service composition to the use case described in Sec. 2. We will employ the UnitDim Ontology [6] as an explicit knowledge model of the domain of units of measurement.

In this use case we implement a demo application that uses an application-independent Controller to compose five Onto⇔SOA services. The services do not depend on

[6] http://www.atoapps.nl/foodinformatics/NewsItem.asp?NID=7

each other and are not specific to the task addressed in the use case, thus can be reused in various application contexts.

A minimal amount of application-specific logic is contained in the demo application, that acts as a consumer. This logic is available neither to the composed services nor to the Controller. The demo application only supplies the facts describing the initial situations in terms of a mathematical statement and, optionally, units of measurement assigned to the identifiers used by it. The Controller invokes (in a non-predefined order) the following services to work on these descriptions:

- The Parser service (see Sec. 6.1) transforms a mathematical statement [7] into a parse tree that represents its underlying structure. For example, the statement $F = m \times a$; is decomposed into the assignment expression that consists of variable $F$ and the multiplication expression $m \times a$, which in turn includes two variables $m$ and $a$.
- The Unit Assigner service recognizes variable names and automatically assigns units of measurement commonly used for those variables. For example, this service will assign the unit of measurement Newton to variable $F$.
- The Unit Consistency Checker service (see Sec. 6.2) analyzes the structure of a mathematical expression with units of measurement assigned to its variables. This service attaches one of the three possible values (*unit consistent*, *unit inconsistent*, *unit consistency unknown*) to the statement and to each of its sub-expressions.

In the coming subsections we elaborate upon the design of the Parser and Unit Consistency Checker services.

## 6.1 Parser Service

In Onto⇔SOA a service is described via its service ontology only. This service ontology defines the vocabulary needed to express input and output documents sent to and created by the service. The sole task of the Parser service is to transform a given input statement into a tree of expressions. Hence, we have to define a service ontology capable of supporting this task.

The *statement* is the central concept in this service ontology. Each statement is linked to its *source* – a representation of the statement in a Matlab-like language, and to a collection of *expressions* into which the statement is parsed. Each expression in turn also has a source and can be linked to other expressions, thus forming a hierarchical structure. An expression is an abstract type that does not directly appear in the communicated documents. Instead it serves as a super-type for more specific kinds of expressions such as *assignment expression*, *multiplication expression*, *identifier* etc. which do appear in those documents.

If we consider an input document that contains the facts labelled with **`0`** on Fig. 4 then the Parser service will extend this document with facts labelled with **`1-MP`** on Fig. 4. If an input document does not contain the source of a statement, the Parser service cannot infer a corresponding hierarchical structure, thus returning an unmodified document. If an input document already contains the decomposition tree, then the Parser service returns it unmodified because it may not override existing facts.

[7] We support a subset of the syntax of the Matlab language to express mathematical statements from science and engineering. This subset also occurs in many other programming languages.

### 6.2 Unit Consistency Checker Service

The service ontology of the Unit Consistency Checker service combines parts of the service ontologies of the Parser and Unit Assigner services and adds the *unit consistency* property to the concepts Statement and Expression.

During operation, the Unit Consistency Checker service takes an input document like the one produced by the Parser service, with units of measurements assigned by the Unit Assigner service (e.g., a document containing facts labelled with **`0`**, **`1-MP`** and **`2-AU`** on Fig. 4). The service determines consistency of each expression and of the overall statement resulting in the document shown on Fig. 4.

The Unit Consistency Checker service is able to infer unit consistency of a statement or an expression only if the conditions below are met by the input document:

- the statement has been decomposed into sub-expressions;
- units of measurement are assigned to identifiers;
- the input document does not yet contain the consistency of the statement or an expression.

### 6.3 Sample Run

To illustrate the composition process we describe in detail a sample run. In this, run we compose three services: the Unit Consistency Checker (UC), the Unit Assigner (UA) and the Parser (MP). Figure 4 shows a trace of the blackboard during this run. Tags on the left-hand side identify in which iteration and by which service a certain fact has been introduced into the blackboard.

A description of the initial situation in the application domain (a given mathematical statement) is supplied by the user. The demo application expresses it as a document and submits it to the blackboard. On the trace the lines marked with the **0** tag correspond to this state.

**Iteration 1:** The controller starts invoking services (the order corresponds to their appearance in the text). Neither UC nor UA can contribute to the blackboard because it contains no expressions. MP contributes to the blackboard by decomposing the statement into a tree of expressions.

The controller determines that the blackboard has been modified during the first iteration and proceeds with **iteration 2**. UC still cannot contribute to the blackboard because the expressions do not have units of measurement assigned. UA contributes to the blackboard by assigning units of measurement to the identifier expressions (variables). MP adds nothing to the blackboard because the statement has already been decomposed into expressions.

Again the controller determines that the blackboard has been modified and proceeds to **iteration 3**. UC contributes to the blackboard by inferring unit consistency of all expressions and the overall statement. Neither UA nor MP can contribute new facts to the blackboard. The controller determines that the blackboard has been modified and proceeds to **iteration 4** during which neither of the services can contribute to the blackboard because it already contains all the facts. After this iteration the blackboard is not modified, thus the controller stops iterating and sends the final content of the blackboard to the consumer (demo application).

## 7 Discussion

In the described sample run 12 service invocations (4 iterations $\times$ 3 services) took place out of which only 3 were productive. This represents 300% *invocation overhead* and corresponds to the worst-case composition scenario: only one productive invocation in each iteration but the final one during which there are no productive invocations at all. This overhead can be reduced if the Controller can remember an order of productive invocations and re-apply it in future compositions. If such a strategy is employed then our sample composition would require 6 service invocations only (2 iterations $\times$ 3 services) reducing the invocation overhead to 100% caused by the final iteration. The final iteration is required to be completely unproductive to identify the end of the composition.

```
0     Matlab statement
0      has source F = m × a;
3-UC    unit consistency INCONSISTENT
1-MP    contains expressions
1-MP      assignment
1-MP        has source F = m × a
3-UC        unit consistency INCONSISTENT
1-MP        contains expressions
1-MP          identifier
1-MP            has source F
3-UC            unit consistency CONSISTENT
2-UA            has unit Newton
1-MP          multiplication
1-MP            has source m × a
3-UC            unit consistency CONSISTENT
1-MP            contains expressions
1-MP              identifier
1-MP                has source m
3-UC                unit consistency CONSISTENT
2-UA                has unit kilogram
1-MP              identifier
1-MP                has source a
3-UC                unit consistency CONSISTENT
2-UA                has unit metre per second squared
```

**Fig. 4.** A trace of the blackboard during the sample run. Tags on the left-hand side identify in which iteration and by which service (UC - Unit Consistency Checker, UA - Unit Assigner, MP - Parser) a certain fact has been introduced into the blackboard.

This indicates that the proposed composition approach should be applied with extra care in the context where invocation costs are significant. For example, data-intensive services that expect large amount of data to be passed through their interfaces bear significant invocation costs, and thus are likely to be inefficient for the Blackboard-style composition. Though, different design techniques can be applied to reduce size of communicated messages, in Onto⇔SOA the preferred approach would be to re-model a data-centric service on a higher conceptual level 'compressing' operations on raw data into semantically richer notions. For example, instead of *actually converting* arrays of numerical data measured in pounds into kilograms, a service can describe *how* a conversion between these two units of measurement can be performed: by multiplying values by 0.454. This design demonstrates a Knowledge-oriented direction in service design promoted in Onto⇔SOA via ontology-based service interfaces, and further enforced by the Blackboard-based composition mechanism.

In Onto⇔SOA we encourage limiting an architecture to as few services as possible (ideally to one) for a given application domain (a task). This assumes that the application domain is formulated on the acceptable (to a service provider) level of granularity. To achieve this level an initial application domain may need be decomposed into subdomains (sub-tasks) each of them can be implemented in a single service. If we assume that the proposed Blackboard-based composition is used to realize the initial complex task out of its sub-tasks, then the decomposition process can be directed by constraints defined in Onto⇔SOA and the Blackboard-based composition mechanism.

In the proposed composition scenario we assume that it is a responsibility of a consumer to determine what services are required to realize a desirable complex functionality. So far we have not directly addressed the task of service discovery. However, one of the design guidelines defined in Onto⇔SOA is to have a 1-to-1 relationship between a service and its service ontology (and corresponding concepts). If followed, this makes it trivial for a controller to find a service that corresponds to a concept on a blackboard. We will investigate the discovery problem in future work.

A Blackboard-based composition is also applicable to services that do not fit into the Onto⇔SOA service model. However, additional measures are required to resolve potential conflicts resulting from relaxed/missing Onto⇔SOA constraints (data-model, semantic, protocol and dependency related conflicts, concurrency issues etc).

One of the general distinguishing features of our approach is that, unlike in e.g. Semantic Web Services, we do not aim to cover as many composition scenarios as possible with a complex composition framework. Instead, we limit ourselves to a unified, simple yet feasible in practice composition scenario, and identify requirements (provide design guidelines) which must be met for that scenario to work.

Consequently, this limits applicability of our framework primarily to the design of new services. Many existing services, especially RPC-based ones, are not designed to meet the Onto⇔SOA constraints (document-orientation and session statelessness, to name a few), thus cannot be readily composed with the proposed Blackboard-style mechanism.

## 8 Conclusions

We have proposed an approach to service composition inspired by the ideas from Blackboard Systems extensively studied in the AI community. It is believed that Blackboard Systems provide a very flexible and modular architecture for integration of independent coarsely-grained components. We have integrated a Blackboard-style composition into Onto⇔SOA – a SOA design framework that employs a restricted document-oriented and ontology-based service model and provides guidelines for construction of reusable services.

We have devised a Blackboard-style composition mechanism that utilizes an application-independent controller component and a homogeneously structured repository (a blackboard) through which services interact. The proposed approach requires neither extensive service model nor explicit workflow specification and enables composite functionality to emerge by bringing a number of services together and making them interact via a shared data repository.

We have confirmed feasibility of the proposed mechanism by having used it to compose five Onto⇔SOA services in the described units consistency checking use case. We have observed that the proposed Blackboard-style composition fits particularly well to the document-oriented and ontology-based service model. This implies that in many non-trivial application scenarios, such as the described use case, a service ontology (a description of a document-oriented service interface) is sufficient to enable service composition.

## References

1. W3C: OWL-S: Semantic Markup for Web Services. (http://www.w3.org)
2. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. Applied Ontology **1** (2005) 77–106
3. Hakimpour, F., Sell, D., Cabral, L., Domingue, J., Motta, E.: Semantic web service composition in irs-iii: The structured approach. In: CEC '05: Proceedings of the Seventh IEEE International Conference on E-Commerce Technology (CEC'05), Washington, DC, USA, IEEE Computer Society (2005) 484–487
4. Korotkiy, M., Top, J.: Onto-SOA: From Ontology-enabled SOA to Service-enabled Ontologies. In: International Conference on Internet and Web Application and Services (ICIW'06). Guadeloupe. (2006)
5. Korotkiy, M., Top, J.: MoRe Semantic Web Applications. In: Proceedings of the ESWC'05 workshop on User Aspects of the Semantic Web. (2005)
6. Brickley, D., Guha, R.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation. http://www.w3.org/TR/rdf-schema/ (2004)
7. W3C: Resource Description Framework (RDF). (http://www.w3.org/RDF/)
8. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, UNIVERSITY OF CALIFORNIA (2005)
9. Korotkiy, M., Top, J.: Designing a Document Retrieval Service with Onto-SOA. In: Proceedings of the Semantic Web Enabled Software Engineering workshop at ISWC'06. (2006)
10. Engelmore, R., Morgan, T., eds.: Blackboard Systems. Addison-Wesley (1988)
11. Corkill, D.: Blackboard Systems. AI Expert **6** (1991)
12. Pfleger, K., Hayes-Roth, B.: An Introduction to Blackboard-style Systems Organization. Technical report, Stanford University (1997)