Ontology-Based Software Analysis and Reengineering Tool Integration: The OASIS Service-Sharing Methodology

Dean Jin
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada
djin@cs.umanitoba.ca

James R. Cordy
School of Computing
Queen's University
Kingston, Ontario, Canada
cordy@cs.queensu.ca

Abstract

A common and difficult maintenance activity is the integration of existing software components or tools into a consistent and interoperable whole. One area in which this has proven particularly difficult is in the domain of software analysis and reengineering tools, which have a very poor record of interoperability. This paper outlines our experience in facilitating tool integration using a service-sharing methodology that employs a domain ontology and specially constructed, external tool adapters. A proof of concept implementation among three tools allowed us to explore service-sharing as a viable means for facilitating interoperability among these tools.

1. Introduction

The software reengineering community has actively responded to the needs of maintenance practitioners involved in program comprehension and software analysis. Many tools that provide assistance in carrying out reengineering tasks have been developed. Each of these tools typically provides a specific, specialized functionality to software practitioners [4, 2]. While they are effective operating as independent systems, the usefulness of these tools cannot be maximized without the ability for them to interoperate with other tools [1, 5, 6]. Creation of a suite of tools to support software reengineering requires a means for sharing the services each tool provides among other tools participating in an integration environment.

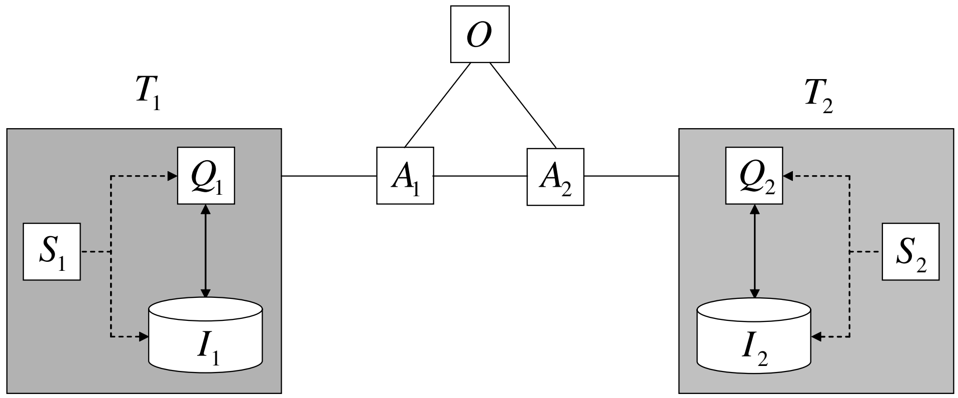
The *Ontological Adaptive Service-Sharing Integration System (OASIS)* is a novel approach to integration that makes use of specially constructed, external tool adapters and a domain ontology to facilitate software reengineering tool interoperability through service-sharing. This short paper provides an outline of the design and function of OASIS.

2. Data vs. Service-Sharing Integration

While the motivation to interoperate software reengineering tools remains strong, very little progress towards achieving this goal has been made. Previous approaches to reengineering tool integration have been *data centric*, concentrating on the exchange of data through specialized hard-coded interfaces (APIs) or rigid standardized exchange formats. The main problem with these approaches is that they are *prescriptive*. They force tool developers to provide a particular functionality to another tool or conform to an idiomatic standard to participate in the integration process.

For developers who integrate their tools, all the time and effort spent generally yields a solution that is specialized for a single tool-to-tool application. Transforming the syntactic and semantic information represented in one tool to a form that is compatible with another tool is tedious and time-consuming, and the effort expended is largely lost when developers want to integrate with other tools. For n tools that want to interoperate, the transformation process must be repeated $n-1$ times. As a consequence, data centric integration has left software maintainers with a broad range of autonomous tools that do not work effectively with other tools. Linkages among tools that do interoperate are hard to unravel and difficult to generalize for use in an open integration environment. This makes it difficult for other tool developers to participate in the integration process.

We advocate *non-prescriptive* integration, focusing on sharing the *services* offered by each tool rather than simply exchanging data among them. A tool service is the functionality provided by a tool that, when given a set of one or more inputs, generates a corresponding output that is relevant to software maintainers. In the case of reengineering tools, the inputs are typically source code (or facts derived or inferred from source code) and the output is typically a report or visualization. Often a tool will provide more than one tool service.



A_x = Conceptual Service Adapter I_x = Factbase Instance O = Domain Ontology
Q_x = Transaction Set S_x = Schema T_x = Tool Participant

Figure 1. The OASIS Architecture

3. The OASIS Architecture

OASIS is an integration methodology that provides a means for reengineering tools to work cooperatively to share services and assist maintainers in carrying out software analysis and program comprehension tasks.

Consider two or more reengineering tools that we want to cooperate in an *integration*. Here we use the term integration to define the environmental boundaries (i.e. the set of tools) that OASIS will operate between. A tool in the integration is referred to as a *participant*.

Each participant offers a set of services to the integration that are shared with the other participants. It is not necessary to share all the services offered by a participant in the integration, although at least one service must be shared (otherwise there is no reason for the tool to participate). Note that a tool that only supplies a factbase is in fact providing a service, namely one of representing software facts extracted from source code in a particular structured format.

Figure 1 provides an architectural view of OASIS. To keep things simple, we show only two tools (T_1 and T_2) involved in an integration. An actual OASIS integration can have any number of participants. Each of the participant tools (T_1 and T_2) provides a set of transactions (Q_1 and Q_2), a schema (S_1 and S_2) and a correspondingly structured factbase instance (I_1 and I_2).

Within each tool, a directed, dashed line reflects the important role the schema plays in defining the representation supported by the instance and the structure of the transactions that operate on the instance. A solid, bidirectional line indicates the close operative relationship the transactions have on the instance.

The OASIS methodology involves the creation of two sets of components:

1. **Domain Ontology** (O). This component stores all the knowledge required to support service-sharing among each of the tools participating in the integration. The knowledge is stored as a tabularized, cross-referenced compilation of representational concepts and services offered by each integration participant.

   Taken together, the representational concepts stored in the domain ontology define a *conceptual space*, consisting of conceptual 'slots' that fact instances fit into. A fact instance fits into a slot only when the concept it represents matches a concept in the domain ontology. We say that a tool has *concept support* when this occurs. We describe concept support in more detail in an earlier paper [3]. Shared services only operate on fact instances that actually fit into these conceptual slots.

   A service offered by a tool participating in an OASIS integration can be shared only when the concepts required by the service intersect with the concepts supported by another participant tool.

   Each OASIS implementation requires only one domain ontology.

2. **Conceptual Service Adapters** (A_1, A_2). These components function as integration facilitators for tools participating in the integration. In an OASIS implementation, each tool is affiliated with a single conceptual service adapter. Each makes use of the domain

ontology to get the information it needs to regulate the integration process.

Conceptual service adapters perform the following three main functions:

(a) *Shared Service and Concept Support Identification.* Making use of the knowledge stored in the domain ontology, each conceptual service adapter identifies requests for shared services and determines the concepts each service requires.

(b) *Factbase Filtering.* Depending on the mode of operation invoked, each conceptual service adapter will map all fact instances into and out of the conceptual space defined by the domain ontology. This process is known as *filtering*. Mapping fact instances into the conceptual space is performed by an *inFilter*. Mapping from the conceptual space is performed by an *outFilter*. Both of these filters are specially tailored to work with the representation supported by the factbase for the tool that the conceptual service adapter is associated with.

(c) *Shared Service Execution.* Each conceptual service adapter manages requests from other conceptual service adapters for the execution of shared services on the tool they are associated with.

Although all the conceptual service adapters have the same basic architecture and operating characteristics, each is specially constructed to handle the functional and information filtering aspects of it's corresponding tool that are required to facilitate interoperability.

The communication links between the domain ontology, the conceptual service adapters, and tools they are associated with are shown as solid black lines in Figure 1.

4. How OASIS Works

In order to show how an OASIS implementation works, consider the two reengineering tools T_1 and T_2 as shown in Figure 1. This is the base case for our integration paradigm. An OASIS implementation can have any number of participants. We only show two here to keep the explanation on how OASIS works as simple as possible.

The goal of our integration effort is to apply a service available in one participant to the factbase of another participant. In this example, T_2 offers a service V consisting of the sequential application of transactions q_x, q_y and q_z (a subset of the complete set of transactions offered by Q_2). We would like to apply service V to I_1, the factbase for T_1. This has the effect of sharing service V with T_1.

The domain ontology O has been constructed and contains knowledge of the representational concepts and services supported by tools T_1 and T_2. The conceptual service adapters A_1 and A_2 facilitate the interoperability we need to achieve our goal.

The flow of information through the OASIS components is shown for each of the following steps in Figure 2.

(1) We start with I_1. The user calls A_1 and requests service V. Adapter A_1 uses the ontology to identify V as a service offered by T_2. It also learns that V requires a factbase that supports (in this example) three concepts known as c_1, c_2 and c_3 in the ontology. The factbase for T_1 must support the concepts that service V operates on. A_1 accesses the ontology for a second time and verifies that T_1 supports a representation for concepts c_1, c_2 and c_3. If a tool does not support the required concept(s), the integration attempt will terminate at this point.

(2) A_1 invokes it's inFilter to map all fact instances from the I_1 factbase into the conceptual space. A_1 then sends a request to A_2 asking it to execute service V.

(3) Acting on the request from A_1, A_2 invokes it's outFilter to map the conceptual space representation to I_{temp}, a local factbase instance for T_2.

(4) A_2 then instructs T_2 to apply service V to I_{temp}. This produces results in the new T_2 factbase I_{result}.

(5) A_2 invokes it's inFilter to map the results in I_{result} to the conceptual space. It then sends a message to A_1 indicating the service has completed.

(6) Acting on the message from A_2, A_1 invokes it's outFilter to map the conceptual space representation to I_{result}, a local factbase for T_1. The completed integration terminates.

In this example, service V is essentially *shared*; it can be applied to fact instances from I_1 and I_2. Any reengineering tool that supports concepts c_1, c_2 and c_3 can share service V from T_2 in this manner.

Using Figure 1 as a reference, the effect of service-sharing can be indicated by a solid bidirectional line stemming from a set of transactions from one tool (Q_x) to a factbase instance of another tool (I_y).

This tiny example serves to demonstrate the technique. In practice we have demonstrated integration between shared services offered by three tools in our OASIS proof-of-concept implementation. We applied our OASIS implementation towards the analysis of production-sized software systems including Linux Kernel v2.0.27a (14,338 source facts), the TXL language processor v6.0 (9,000 Turing+ LOC, 6,780 design facts) and IBM's Tobey code generation system (250,000 PLIX LOC, 11,066 architecture facts).

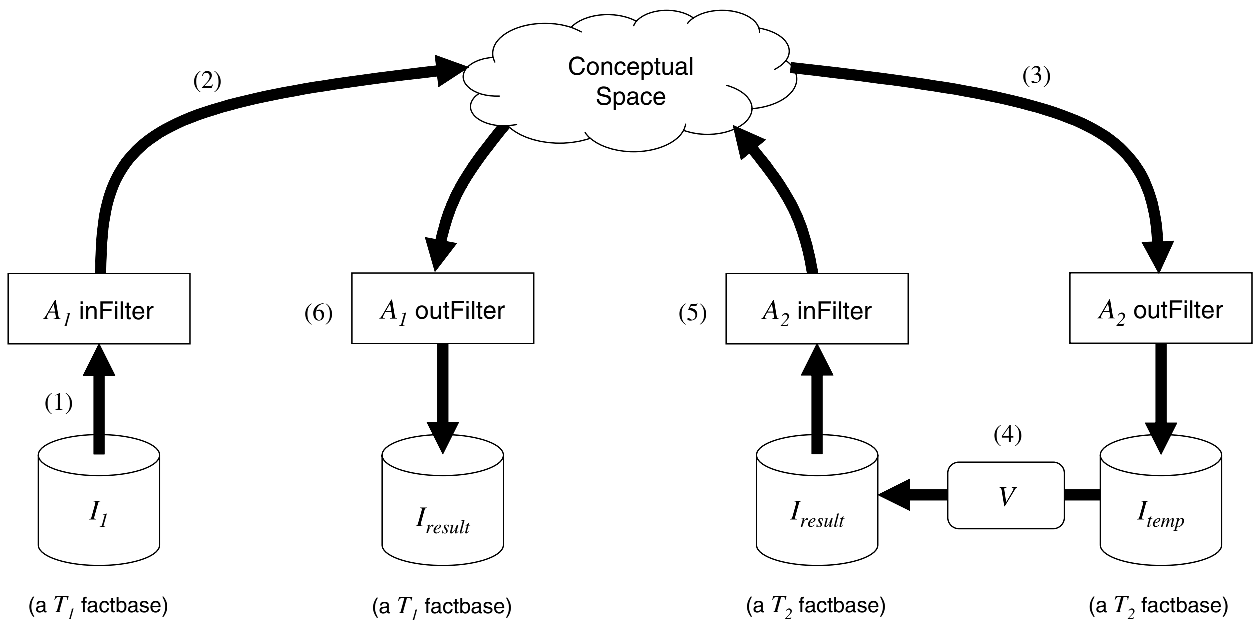


Figure 2. Sharing Service V with Tool T_1

5. Discussion and Conclusion

One of the major pitfalls of previous attempts to facilitate integration among reengineering tools has been the prescriptive methodologies that tool developers have been forced to work with. The primary goal of OASIS is to simplify the work involved in participating in the integration process. We believe that this can be accomplished by maintaining a clear separation between each participant and the components that look after the complexities of integration. This provides the following advantages:

Independence. Each integration participant works in an independent manner with no dependencies among them except for those that relate to either of the interfaces that OASIS provides.

Changeability. Any participant can be changed without affecting the integration as long as the changes do not affect any of the interfaces provided by OASIS.

Localization. Maintaining the participants is made easier through the separation of concerns that exists among them. Any implementation problem in a participant remains localized unless it affects either the conceptual or operational interfaces offered by OASIS.

OASIS provides a conceptual interface (the domain ontology) and operational interfaces (conceptual transaction adapters) to each participant that facilitate integration without revealing the details of their implementation.

This work was supported by Natural Sciences and Engineering Research Council of Canada. We thank the anonymous referees for their helpful and useful suggestions for improving the original paper, and apologize that the restrictions on a short paper make it difficult to act on them in any meaningful way.

References

[1] J. Ebert, B. Kullbach, and A. Winter. "GraX – An Interchange Format for Reengineering Tools". In *Proceedings of WCRE'99*, pages 89–98, 1999.

[2] G. Y. Guo, J. M. Atlee, and R. Kazman. "A Software Architecture Reconstruction Method". In *Software Architecture*, pages 15–33. Kluwer Academic Publisher, February 1999.

[3] D. Jin, J. R. Cordy, and T. R. Dean. "Transparent Reverse Engineering Tool Integration Using a Conceptual Transaction Adapter". In *Proceedings of the 7th European Conference on Software Maintenance and Reengineering (CSMR 2003)*, pages 399–408, Benevento, Italy, March 2003.

[4] T. C. Lethbridge. Requirements and Proposal for a Software Information Exchange Format (SIEF) Standard. Draft Manuscript, November 21 1998. URL: `http://www.site.uottawa.ca/~tcl/papers/sief/standardProposal.html`.

[5] S. Perelgut. "The Case for a Single Data Exchange Format". In *Proceedings of WCRE'00*, Nov. 2000.

[6] S. Woods, L. O'Brien, T. Lin, K. Gallagher, and A. Quilici. "An Architecture For Interoperable Program Understanding Tools". In *Proceedings of the 6th International Workshop on Program Comprehension (IWPC'98)*, pages 54–63, Ischia, Italy, June 1998.