Compositional Business-Task Organization

HASIDA Kôiti, IZUMI Noriaki, and MORI Akira
ITRI, AIST
{hasida.k, n.izumi, a-mori}@aist.go.jp

1. Introduction

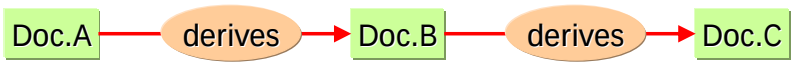
CBTO (Compositional Business-Task Organization) is a declarative framework for designing, coordinating, and operating semantic Web services. In CBTO, the execution of a business task is viewed as the composition of a proof tree, and services are regarded as providing parts to this proof tree. Constraints are described as Horn-clause programs. Each Horn clause is instantiated either interactively following the human user's operation or automatically as part of a program transformation. This declarative approach addresses a simple, user-friendly framework for semantic Web service choreography.

2. Constraint-Based Workflow

People and computers do not share meaning of programs because programs are hard for people to understand. Programs are hard to understand because they are complex. This complexity mainly arises due to descriptions of procedures. So the complexity of programs drastically decreases by abstracting procedures away.

A program without procedural specifications is called a constraint. A constraint program statically describes conditions on some objects, but does not stipulate any procedure to meet those conditions. Compared with procedural programs, constraint programs account for more complex computational processes by simpler specification.

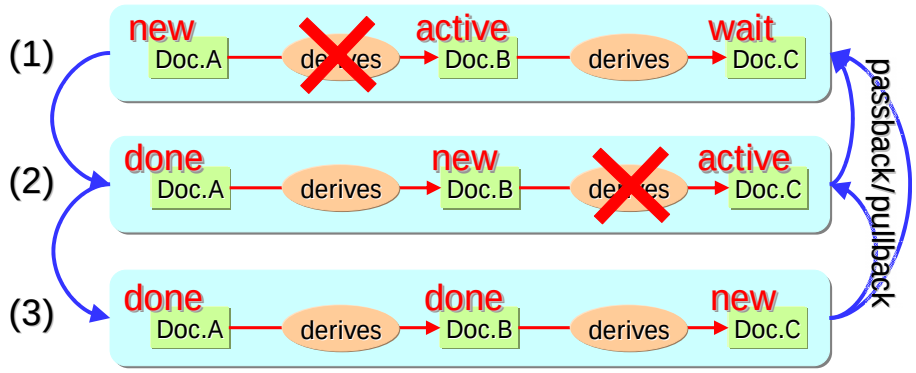
The following figure is a part of CBTO script, which is a constraint program realizing a workflow involving three tasks: the composition of document A, B, and C. The internal structures of the documents are omitted in the figure.



Binary relation `derives' is a constraint to the effect that its second argument is newer than its first argument. Therefore, when document A is finished, the composition of document B is requested, and when document B is done, the composition of document C is requested. Suppose for instance that document A has just been composed. Then it is

newer than document B, which violates the constraint, so that a renewal of document B is requested. This request is implemented by notifying the person in charge of document B that she should work on it.

This constraint program does not necessarily mean that document A, B, and C should be worked on in this fixed order. Document A may be modified when Document C is being worked on after document A and B are done. This violates again the constraint that document B should be newer than document A, and hence a request will be issued to revise document B. Here the person in charge of document A is regarded as having pulled her task back to document A or having passed the work back to document B. Thus the script accounts for very complex processes involving lots of passbacks and pullbacks. In summary, the three steps (1), (2), and (3) are not only followed in this order as indicated by the arrows in the left-hand side of the following figure, but also in other ways as indicated by the arrows in the right-hand side.



3. Interactive Services

Another feature of CBTO is that its constraint-based coordination can accommodate not only one-shot services but also interactive ones, including both ordinary client-server interactions and constraint-satisfaction interactions. For instance, the business interaction involving passbacks and pullbacks discussed above is of a constraint-satisfaction type. OWL-S, WSMO, and other frameworks of Web services, semantic or not, fail to directly address such interactions as far as they assume rather straightforward grounding on WSDL. Unfortunately, OWL-S and WSMO are fundamentally incompatible with interactive services due to their procedural and/or plan-based formulations.

In this connection, the functionality required for CBTO which most of the

constraint-satisfaction systems lack is the treatment of assumptions and contradictions for the sake of interactions with the environment involving human users. As discussed regarding the workflow case, for example, there may arise contradictions during constraint satisfaction. The assumptions containing such contradictions must be maintained rather than rejected as long as these contradictions may be dissolved through some interactions with the environment.

4. Operational Issues

Built-in predicates and demons are used to implement constraints which are hard or impossible to address by Horn clauses. For instance, the `derives' property discussed above is a built-in predicate. The computation to obtain eigenvalues of matrices should be implemented as a built-in predicate, too. The constraint that one same person cannot participate in two different events at one time can be realized as a demon triggered when somebody is involved in two different events on, say, the same day. Implemented as another demon is the constraint to the effect that two events at different places involving the same person must accompany an event of her transportation in between.

Many built-in predicates restrict the input/output directions of their arguments. For instance, it is unrealistic to reverse the I/O of the computation of matrix eigenvalues. Predicates defined by Horn clauses may also be directed if some of their definition clauses contain directed predicates. To guarantee that no deadlock occurs in CBTO programs, we can check that each Horn clause contains no fixed I/O cycle, assuming that each argument of each predicate has a fixed I/O pattern; i.e., an argument is either fixed for input only, for output only, or for both.

5. Concluding Remarks

Introducing external Web services as atomic formulae (literals) with externally defined predicates, CBTO provides a light-weight and simple infrastructure for semantic Web service to coordinate various services based on their meaning. In this connection, various types of groupware including project management systems and enterprise systems can be implemented by combining CBTO with GUI generated by ontology-based stylesheets, though we do not discuss further details here.

Several frameworks of semantic Web services have been proposed. OWL-S is close to traditional procedural programming language. WSMO is a framework based on the planning technology in AI. WSMO may be regarded as a constraint-based method, but WSMO programs tend to be more complicated than CBTO programs because in WSMO

you must specify preconditions and postconditions of each action. Rule language WRL, which is presupposed by OWL-S and WSMO, involves Horn clauses. So CBTO is a subset of WRL and thus both its specification and its programs are much simpler than in OWL-S, WSMO, and even WRL.