Semantics and Provenance for Processing Element Composition in Dispel Workflows

Eric Griffis
UCLA
Los Angeles, CA 90095
eric@emdti.com

Paul Martin
University of Edinburgh
Edinburgh EH8 9AB
pmartin@staffmail.ed.ac.uk

James Cheney
University of Edinburgh
Edinburgh EH8 9AB
jcheney@staffmail.ed.ac.uk

1 Introduction

There have been a great number of workflow management systems developed specifically for e-science applications [2]. Many graphical workflow systems exist in the context of a number of other frameworks for large-scale scientific data processing. Though graphical composition has many advantages, the visual construction of workflows can become cumbersome as the structural complexity of those workflows increases. Many of these systems have at some point developed their own description languages such as MOML (Kepler) and SCUFL (Taverna). These languages are generally declarative, designed to be machine-readable, and typically generated via some graphical interface. However, such interfaces typically do not support schematic definitions of workflows, i.e. generic workflows specifying computations at a higher level of abstraction than regular workflows. Thus, there exists a usecase for human-writable workflow composition languages that use conventional programming constructs to succinctly describe the construction of workflows such that a workflow description can be generated by executing a simple script. Human-writable workflow composition languages already exist, but are usually strongly tied to a particular data processing framework.

Dispel is a workflow composition language intended for distributed, data-intensive, streaming-pipeline applications [1]. It is designed to insulate researchers from the underlying computational middleware used in such applications by relying on generic components published in an open repository. Applications are modeled as pipelines of persistent processing elements that consume and produce data streams [3]; the composition of these elements and the data-flows between them determine the behavior of the system as a whole.

While it has much in common with other (graphical or textual) workflow languages mentioned above, Dispel's distinctive feature is the capability to use programming features to construct families of workflows. The key aspect of Dispel that makes this possible is the capability to define *composite processing elements* (or composite PEs), that is, subroutines that construct subgraphs of the final workflow graph. Among other applications, composite PEs can be used to define generic libraries of pipelined operations, to abstract over families of graph structures, and to wrap functionality such as provenance tracking [4].

2 Implementation

To increase confidence in the correctness of our formal semantics, we implemented an interpreter in Racket [1], a programming language similar to Scheme. We translated each rule directly into a Racket function. Dispel also has two existing implementations (with a common core library): the main implementation is a Web gateway that accepts Dispel programs, communicates with a registry and repository to obtain PE implementations, and submits the resulting job to the OGSA-DAI distributed enactment engine. The second implementation, Dispel-Lite, is a lightweight wrapper to the main Dispel parser and interpreter that requires all PE implementations to be provided locally, and runs them locally.

Our Racket implementation was developed as a way of testing assumptions made in our semantics against the actual behavior of Dispel (in practice, we primarily compared with Dispel-Lite, which has some known differences in behavior from the main Dispel implementation). We compared the behavior

[1] http://racket-lang.org/

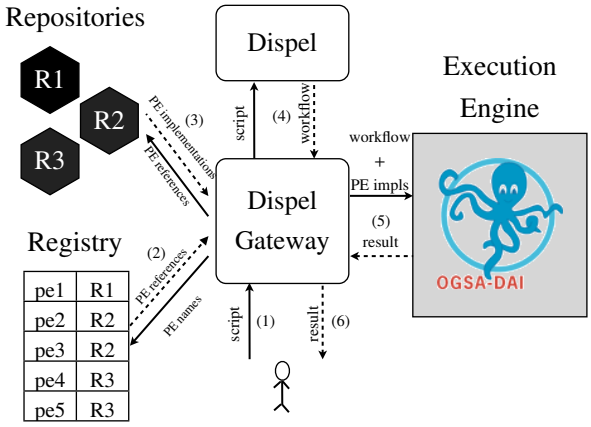

Figure 1: Dispel architecture. In a typical run, a user (1) submits a script to the gateway, which (2) looks up the primitive PEs referenced in the script in the registry and (3) obtains their implementations from the repositories. The script is evaluated (4) to a workflow, and (5) the workflow and PE implementations are submitted the job to the computational back-end for execution; the end result (6) is made available to the user.

predicted by the semantics with the Dispel-Lite implementation on several small examples. The lessons learned from these comparisons are already incorporated into the operational semantics rules, particularly those having to do with port references, linking statements, and the different stages of PE initialization and composition.

3 Conclusion

We have presented a formal semantics for a core subset of Dispel, a scripting language for defining workflows. Dispel supports some distinctive features, such as processing element composition, that are not described formally in any previous work to the best of our knowledge. Although we have focused on a sublanguage and have not yet extended the semantics to handle all features of Dispel, the remaining features (arrays, records, etc.) are mostly standard or straightforward. We also discussed an implementation we have developed to compare the semantics with the actual behavior of Dispel programs, and shown how to adapt the semantics to partially support simple forms of provenance tracking. This paper represents work in progress and the immediate next steps are to flesh out (and prove correctness for) the provenance semantics, and experiment with techniques for recording provenance at different levels of granularity based on user preferences.

References

[1] Malcolm Atkinson, Chee Sun Liew, Michelle Galea, Paul Martin, Amrey Krause, Adrian Mouat, Oscar Corcho, and David Snelling. Data-intensive architecture for scientific knowledge discovery. *Distributed and Parallel Databases*, 30(5-6):307–324, 2012.

[2] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[3] Liangxiu Han, Chee Sun Liew, Jano Van Hemert, and Malcolm Atkinson. A generic parallel processing model for facilitating data mining and integration. *Parallel Computing*, 37(3):157–171, 2011.

[4] Alessandro Spinuso, James Cheney, and Malcolm P. Atkinson. Provenance for seismological processing pipelines in a distributed streaming workflow. In Giovanna Guerrini, editor, *EDBT/ICDT Workshops*, pages 307–312. ACM, 2013.