Planning-Based Composition of Stream Processing Applications

Mark D. Feblowitz, Anand Ranganathan, Anton V. Riabov, and **Octavian Udrea**
IBM T. J. Watson Research Center
PO Box 704, Yorktown Heights, NY 10598, USA

Abstract

In this demonstration we present our planning-based tools for software composition, and in particular, for composition of distributed stream processing applications. The applications composed by our tools are deployed on IBM InfoSphere Streams distributed stream processing middleware. The applications are used to process large data volumes in real time on large clusters of commodity servers. Our tools include MARIO, the goal-driven planning-based application composition tool for business users, and an Eclipse-based integrated development environment (IDE) for developing planning domain descriptions.

Overview

Distributed stream processing middleware, such as IBM InfoSphere Streams (IBM InfoSphere Streams), allows software developers to apply the full computational power of numerous commodity servers toward large-scale real-time analysis of streaming data. Recently, applications that place high demands on the rate and volume of input data have emerged in telecommunications, finance, health care, and other industries. Stream processing applications are developed by specifying, in a programming language, the data flows between inputs and outputs of stream processing operators.

To significantly shorten the development cycle and make applications more flexible and responsive to changing business needs, we have developed a planning-based approach that gives business users the ability to assemble stream processing applications for their needs on the fly, without programming or using drag-and-drop interfaces. Similarly to planning-based approaches that were proposed for Web Service composition, e.g. (Traverso and Pistore 2004), we have extended AI planning formulations and techniques to address the problem of composing stream processing applications.

To further simplify the goal-driven composition process, we have developed an iterative goal refinement approach that gives business users the ability to express their goals and explore alternative compositions by choosing from a set of business-relevant terms. The built-in optimization engine helps assemble the best match even for ambiguous goals given by the end users. Developers, on the other hand, can easily describe and publish new operators and data sources for use in new compositions. Our tools, including the web-based goal-driven application composition tool (MARIO), and the planning domain development environment (IDE), have been used in a customer pilot since 2009.

MARIO: A Goal-Driven Application Composition Tool

We have developed MARIO (Bouillet et al. 2008) as a framework for automated composition of analysis flows, with the primary motivation to compose stream processing applications. We have further generalized MARIO to compose and deploy analysis flows on a variety of other platforms, including Web Services, Enterprise Service Bus, and others. MARIO can be extended to support new platforms by adding new plug-ins that generate code and deploy composed flows.

An overview of MARIO and its interactions with business users and other systems is shown graphically in Figure 1. MARIO interacts with the business users via a Web-based interface to receive and refine goals, generates and deploys application code for user-specified goals, and presents the results of execution back to the user.

In MARIO, composition goals are specified as sets of business-relevant keywords (i.e., tags). For example, a financial analyst may request an application that identifies stocks sold below volume-weighted average price (VWAP), and therefore constituting a bargain, by selecting tags "*VWAP, BargainIndex*" as a goal.

A unique feature of MARIO is interactive goal refinement. Possible refinements of the specified goal are generated by the MARIO planner together with the optimal plan, based on the analysis of alternative plans. For example, the planner may suggest *LinearIndex* and *ExponentialIndex* as possible refinements of "*VWAP, BargainIndex*", if there are alternative plans matching to those tags. The users are allowed to specify ambigous goals, allowing the planner to make remaining decisions based on its optimality criteria. On the other hand, the users can also refine goals until there are no more refinements.

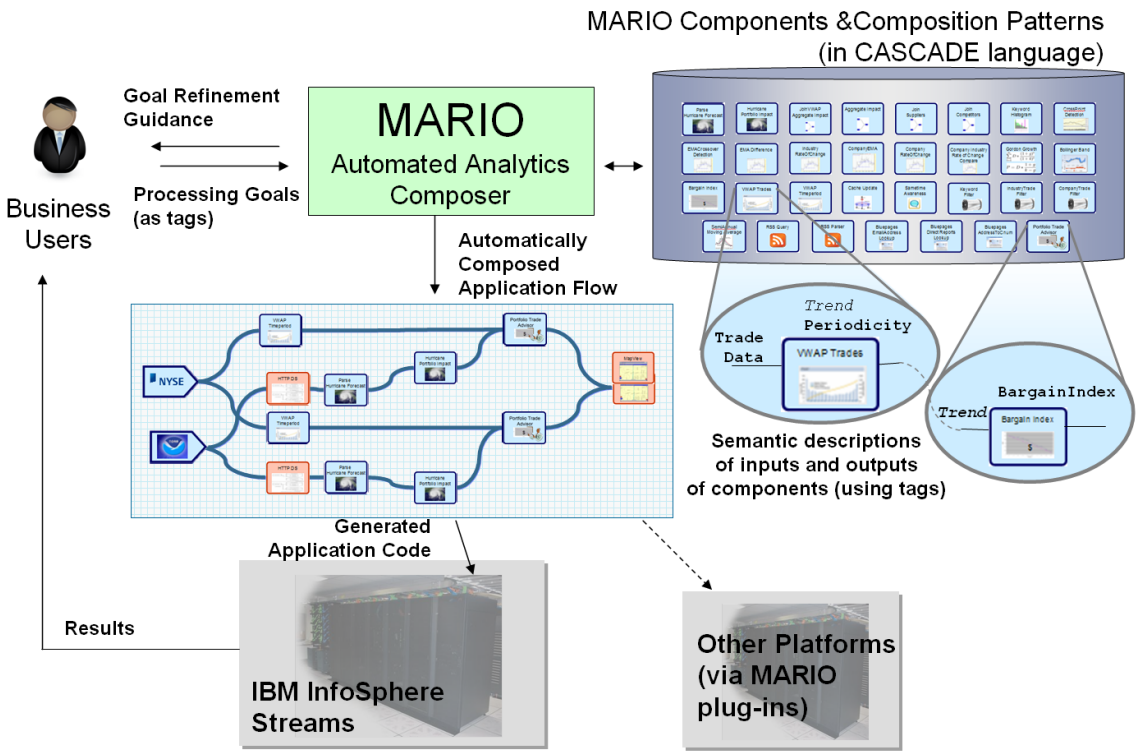

Figure 1: MARIO Overview

In general, goal tags match tags used by the developers in describing inputs and outputs of stream processing operators and data sources. The developers can define sub-tag relationships between tags, allowing MARIO to use reasoning during planning, and match specific annotation tags to general goal tags. For example, tags *LinearIndex* and *ExponentialIndex* can be declared to be sub-tags of *BargainIndex*, and, as a result, the plans that match either tag will also match any goal that includes *BargainIndex*.

In addition to composing applications, which we consider its primary functionality, MARIO Web server includes supporting functions. For business users, it is the primary console for managing and deploying stream processing applications, providing the user interface and server-side functionality for managing long-running applications in a multi-user environment.

Planning Approach

To respond to composition requests efficiently, MARIO relies on a fast special-purpose optimizing planner that solves planning problems described in SPPL domain description language (Riabov and Liu 2006). SPPL descriptions are automatically generated from user-specified goals and developer-defined application composition domains.

In MARIO, tags are used for describing the semantics of analytics and data sources, as well as for specifying the goals. In the past we have implemented a more general OWL-based model (Liu, Ranganathan, and Riabov 2007). However, the OWL-based model was difficult to use in practice, since few stream processing application developers were familiar with OWL. Our current tag-based model is designed to be easier to understand and use, while preserving a few basic reasoning capabilities, such as the sub-tag relationship.

The developers use the Cascade language (Ranganathan, Riabov, and Udrea 2009) to describe application composition domains. For example, the output of a stream processing operator computing exponential index can be tagged with *ExponentialIndex*, and the operator itself can be placed, as one of the possible choices, within a Cascade composition pattern flow graph.

The goals specified by the business users as tags are automatically translated to SPPL problem descriptions. Since goal refinement is the only method for specifying goals, only valid goals can be specified, and only known tags can be included in goals, making this translation simple. Cascade composition patterns and tag relationships are also compiled into SPPL domain descriptions for planning.

MARIO finds an optimal plan and analyzes alternatives every time a user adds or removes a goal tag. This allows users to explore the space of possible plans. However, to make goal refinement process truly interactive, and to serve multiple users simultaneously, planning must be efficient. Our SPPL planner (Riabov and Liu 2006) implements multiple performance optimizations, and achieves planning times of a few seconds for most practical applications.

Goal Tags	Planning Time (seconds)
$\emptyset$	0.11
VWAP	0.16
VWAP, BargainIndex	0.09
VWAP, BargainIndex, ExponentialIndex	0.10
VWAP, BargainIndex, ExponentialIndex, TableView	0.09
VWAP, BargainIndex, ExponentialIndex, TableView, TcpSource	0.09
TcpSource	0.12
TcpSource, TableView	0.08

Table 1: SPPL planner response times, for a sample domain.

Table 1 illustrates the performance of our SPPL planner in MARIO domains. We have measured planning times for several selected goals within a sample application domain (i.e., stock price analysis). Typically, the users will refine goals by selecting one tag a time from possible refinements, starting with an empty goal. In Table 1 we show planning times for two refinement sessions. The SPPL planner was running on a 64-bit Linux node with a quad-core 2.93Ghz Intel Xeon processor with 32GB RAM, however only a single core was used, and the memory usage was below 2GB.

Cascade IDE: A Development Environment for Composition Domains

Cascade IDE is a set of Eclipse (Eclipse Foundation) plugins that developers can use to create and maintain Cascade descriptions of application composition domains for MARIO. The IDE can also import existing stream processing code and generate Cascade descriptions. For example, a stream processing operator invocation implementing exponential bargain index computation can be imported, and its output can be annotated with *ExponentialIndex* tag.

The integration with Eclipse has made it possible to develop a full-featured Cascade editor with syntax highlighting, auto-completion and refactoring. Refactoring allows, for example, to rename a tag, In addition to the editor for Cascade, the IDE includes editors for Web UI configuration and tag relationships.

The planning domains described in Cascade can be deployed directly from the IDE to a MARIO server. Then, double-clicking on a server entry in the IDE launches a browser showing the Web interface. This automation significantly reduces the time spent on routine tasks during the development and testing of application composition domains.

Among many challenges associated with making automated software composition practical, perhaps the most significant is the need to create tools that help developers simultaneously debug a large family of applications generated from a single Cascade project. To address this problem, we have integrated automated test generation and execution techniques with our Cascade IDE (Winbladh and Ranganathan 2011).

Conclusion

We have developed MARIO, a goal-driven tool for automated composition of stream processing applications. This tool allows business users to create applications for their goals without programming. We have also built an Eclipse-based IDE for developers, which allows to describe composable components, such as stream processing operators and data sources, as well as composition patterns, and to make these components and patterns available for application composition by the business users. Our tools have been used in a customer pilot since 2009, and are currently being extended to support other target platforms in addition to IBM InfoSphere Streams.

Acknowledgements and Credits

This work, which has begun eight years ago, would not be possible without the support of our customers, and the contributions from our colleagues and interns at IBM Research. Our former colleagues and team members Zhen Liu, Eric Bouillet, and Hanhua Feng have each contributed to MARIO very significantly. The authors also thank Kristina Winbladh, Shirin Sohrabi and Genady Grabarnik for their contributions. We thank Nagui Halim for continued discussions and valuable feedback. Finally, we thank the anonymous reviewers of our papers who helped improve our work.

References

Bouillet, E.; Feblowitz, M.; Liu, Z.; Ranganathan, A.; and Riabov, A. 2008. A tag-based approach for the design and composition of information processing applications. In *OOPSLA*, 585–602.

Eclipse Foundation Eclipse Project. http://eclipse.org.

IBM InfoSphere Streams http://www.ibm.com /software/data/infosphere/streams/.

Liu, Z.; Ranganathan, A.; and Riabov, A. 2007. A planning approach for message-oriented semantic web service composition. In *AAAI*, 1389–1394.

Ranganathan, A.; Riabov, A.; and Udrea, O. 2009. Mashup-based information retrieval for domain experts. In *CIKM*, 711–720.

Riabov, A., and Liu, Z. 2006. Scalable planning for distributed stream processing systems. In *ICAPS*.

Traverso, P., and Pistore, M. 2004. Automated composition of semantic web services into executable processes. In *ISWC04*.

Winbladh, K., and Ranganathan, A. 2011. Evaluating test selection strategies for end-user specified flow-based applications. In *ASE*, 400–403.