Experiences with the Model-based Generation of Big Data Pipelines

Holger Eichelberger, Cui Qin, Klaus Schmid

{eichelberger, qin, schmid}@sse.uni-hildesheim.de

Software Systems Engineering
University of Hildesheim

www.sse.uni-hildesheim.de

Motivation

- Background FP7 QualiMaster:
  - Configurable and adaptive data processing infrastructure
  - Real-time financial risk analysis
- Programming applications for Big Data frameworks is complex
- Ideal: Focus on data processing, ignore technical complexity

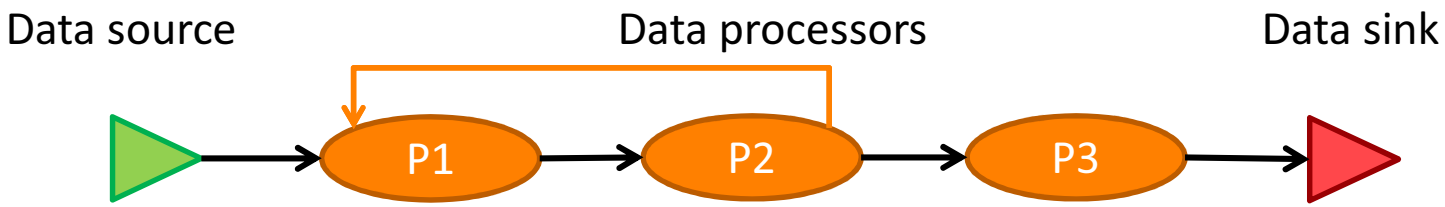
- Goal:
  - Model-based approach to stream processing
  - Hide complexity
  - Ease development
  - Generate complex parts of code
  - Support self-adaptation

Experiences
Lessons learned

Model-based design

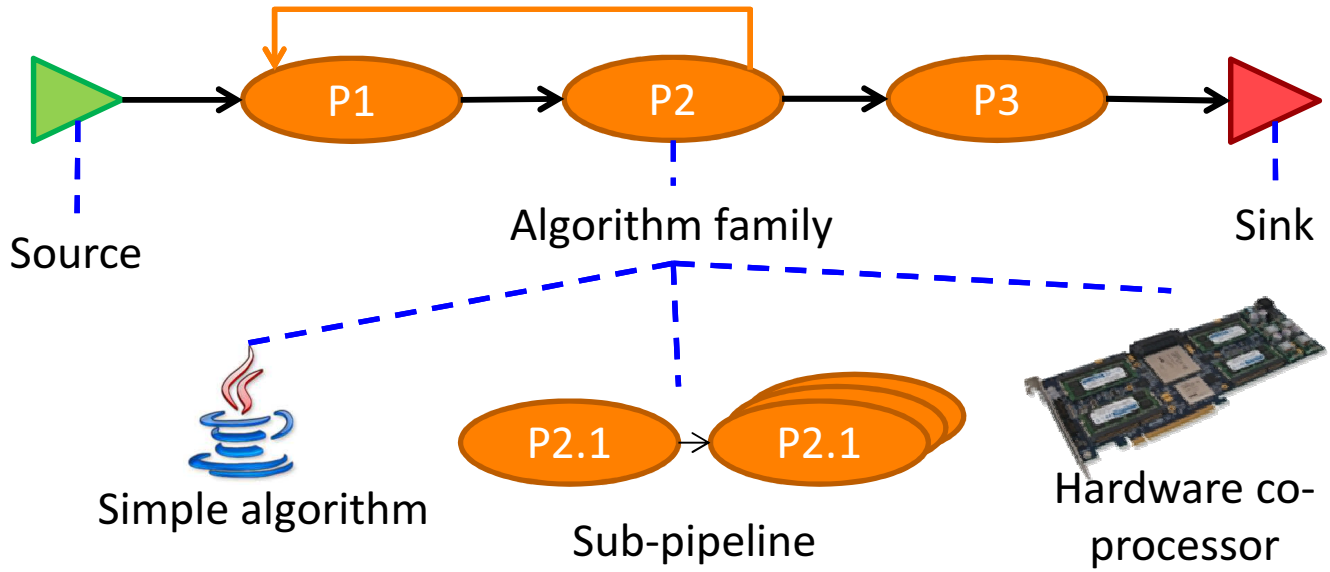
- Basis: Concept analysis
  - Fixed stream operators (e.g., Borealis, PIPES)
  - User-defined operators / algorithms (e.g., Storm, Heron)
  - Combinations (e.g., Spark, Flink)

- Common concept: Data flow graph
  - Typically represented as program
  - Recent trend: DSL



Specific modeling concepts

Data processing pipeline

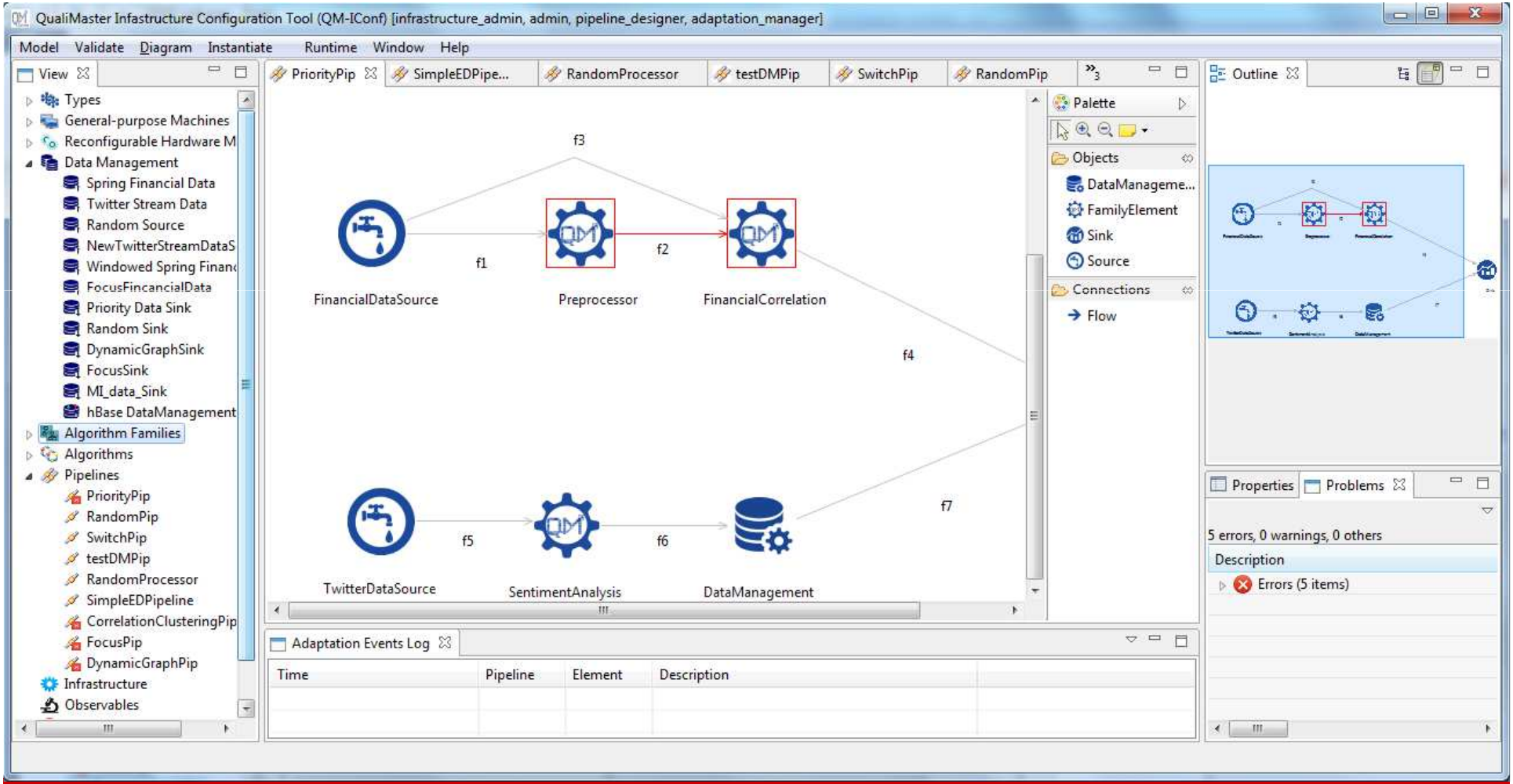


- Domain restrictions
  - Must be a valid data flow graph
  - If $P_s \rightarrow P_e$, P_s must provide types that P_e can process
  - Interface compatibility between families and algorithms

Modeling support

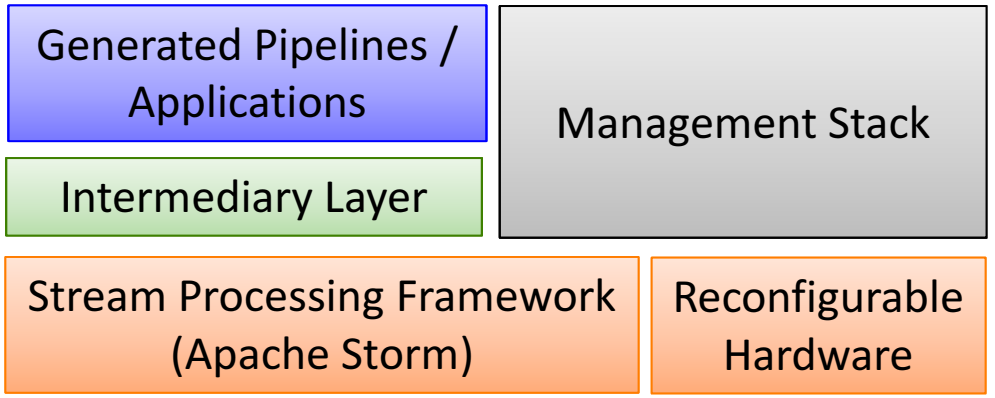
Underlying: Own model-managment framework

Domain-specific modeling frontend



Code generation

- Architecture
  - Heterogeneous resource pool
  - Intermediary layer extending Storm
  - Management layer for runtime
- Generation steps
  - Family interfaces
  - Data serialization support
  - Integration of hardware co-processors
  - Pipelines / sub-pipelines, switching
  - Compile, integrate dependencies, package



16 pipelines
- x 7 code produced
- ~880 MB deployable components

Experiences and Lessons learned (1)

- 7 data engineers from 3 groups, 6 large pipelines
- Beginning of the project
  - Sceptical about model-based approach
  - Initial version after some months
  - Hands-on workshops
  - Feedback:
    - Puzzled about type saftey
    - First own generated pipelines helped
    - Change of focus: More on algorithms
    - Requests for new features, reports on buggy features
- Confidence increased with improved versions (~1 year)

Experiences and Lessons learned (2)

- Later phases
  - Interfaces help to structure work
  - Typing helps avoiding runtime errors
  - “Magic“ of generated code
    - serialization
    - parameters
    - algorithm switching
      - Complex structures due to additional nodes, communication
      - For sub-pipelines: Manual / generated code perform the same
  - Shields from complex coding

Experiences and Lessons learned (2)

- Center of integration ⟶ Higher workload
- Supports evolution
  - Consistent deployment of changes
  - Algorithms must be evolved manually
  - Also errors are deployed easily
- Continuous integration
  - Generation and algorithms
  - Up-to date pipelines are available
  - Intensive tests increase overall build time ⟶ local debugging first
- Effects
  - Focus of work on algorithms
  - Allows realization and evolution of complex structures
  - Avoid runtime issues
  - Stability increases confidence, requires higher quality assurance

Conclusions

- Model-based approach for streaming Big Data applications
  - Type-safe
  - Heterogeneous data processing (hardware co-processors)
  - Flexible exchange of algorithms
- Code generation for Apache Storm
- Approach pays off
  - Positive feedback
  - Requires training, modeling effort, effort for realization of transformation, maintenance and evolution
- Future: Optimized code generation for self-adaptation
  - Switching efficiency
  - Multiple target platforms

} Optimized resource usage is already reality!