Data composition patterns in service-based workflows

Johan Montagnat, I3S team, CNRS
Tristan Glatard, I3S team CNRS/INRIA Asclepios
Diane Lingrand, I3S team, CNRS
Introduction

• **Data parallel applications**
  – Many scientific applications
  – Well suited for exploiting distributed infrastructures
  – Workflow engines ease to transparently exploit parallelism

• **Data composition patterns in workflows**
  – Data-intensive workflows description
  – Expressiveness problem. Trade-off between:
    ▪ Compactness / representation simplicity
    ▪ Flexibility

• **Problem**
  – Define a clear semantics for data composition inside a workflow
Job submission vocabulary

• **Task-based approach**
  – Each job submitted is a **task**
  – Requires a job description language
    ▪ To define: I/O data, executable, command line...
  – Middlewares examples: GLOBUS, LCG2, gLite... batch computing

• **Service-based approach**
  – Each job is a **service**
  – Requires a standard invocation interface (Web Service, GridRPC)
    ▪ Input/Output data are parameters for the service
    ▪ The service is a 'black box' hiding the submission infrastructure
    ▪ Very flexible
  – Example middlewares: DIET, Ninf, Netsolve...
Workflow managers

- **Workflow description**
  - Business workflows (*e.g.* BPEL)
    - Control-centric
  - Scientific workflows (*e.g.* Scufl)
    - Data-centric

- **Workflow execution**
  - **Task-based** workflows (*e.g.* DAGMan)
    - Explicit mention of data dependencies
    - Complex workflow, simple optimisation
  - **Service-based** workflows (*e.g.* Taverna, Triana, Kepler, MOTEUR)
    - Independent expression of processors and input data-sets
    - Simple workflows, complex optimisation

CS friendly

user friendly
Data composition strategies

- Data composition patterns: data intensive applications

**One-to-one**

\[
A = \{ A_1, A_2, A_3 \} = B
\]

\[
A \oplus B = \{ A_1 \oplus B_1, A_2 \oplus B_2, \ldots \}
\]

**All-to-all**

\[
A \otimes B = \{ A_1 \otimes B_1, A_1 \otimes B_2 \ldots A_1 \otimes B_m, A_2 \otimes B_1 \ldots A_2 \otimes B_m, \ldots \}
\]

\[
A \otimes B = \{ A_1 \otimes B_1, A_1 \otimes B_2 \ldots A_1 \otimes B_m, A_2 \otimes B_1 \ldots A_2 \otimes B_m, \ldots \}
\]
Service-based approach versus task-based approach

Graph of services

DAG of tasks

4 data segments
• Explicit priority (parenthesized expression)

\[ A = \{A_0, A_1\} \]
\[ B = \{B_0, B_1\} \]
\[ C = \{C_0, C_1, C_2\} \]

\[ A \oplus (B \otimes C) = \left\{ A_0 \oplus (B_0 \otimes C_0), \quad A_1 \oplus (B_1 \otimes C_0), \right. \]
\[ A_0 \oplus (B_0 \otimes C_1), \quad A_1 \oplus (B_1 \otimes C_1), \]
\[ A_0 \oplus (B_0 \otimes C_2), \quad A_1 \oplus (B_1 \otimes C_2) \right\} \]

\[ B \oplus (A \otimes P) = \left\{ B_0 \oplus (A_0 \otimes P_0), \quad B_1 \oplus (A_1 \otimes P_0), \right. \]
\[ B_0 \oplus (A_0 \otimes P_1), \quad B_1 \oplus (A_1 \otimes P_1), \]
\[ B_0 \oplus (A_0 \otimes P_2), \quad B_1 \oplus (A_1 \otimes P_2) \right\} \]
Data composition in different languages

- **Taverna**
  - One-to-one (dot product) and all-to-all (cross product) operators included in Scufl
  - One-to-one composition results in processing the $\min(\#A, \#B)$ of compositions
  - Based on sequential order

- **Kepler**
  - One-to-one composition
  - Implemented a new actor for all-to-all semantics with the PN director (require work-arounds)

- **Triana**
  - One-to-one composition
MOTEUR workflow manager

- **Efficient workflow enactment**
  - Interfaced to a grid infrastructure (distributed computing)
  - Transparently exploits application (data+service+workflow) parallelism
  - Special emphasis on data-parallelism

- **Includes data composition patterns**
  - Use the Scufl description language
  - Implements the one-to-one composition semantics described in this paper

- **Where?**
  - http://www.i3s.unice.fr/~glatard

- **How?**
  - CeCILL (French-GPL) license
• **Sequential order semantic**

$$\mathbf{B} = \{B_0, B_1\} \quad \mathbf{A} = \{A_0, A_1\} \quad \mathbf{P} = \{P_0, P_1, P_2\}$$

$$\mathbf{B} \oplus_{\text{Taverna}} (\mathbf{A} \otimes \mathbf{P}) = \{ B_0 \oplus (A_0 \otimes P_0), \quad B_1 \oplus (A_1 \otimes P_0) \}$$
Different semantics?

- **Sequential order semantic**

\[ B = \{ B_0, B_1 \} \]
\[ A = \{ A_0, A_1 \} \]
\[ P = \{ P_0, P_1, P_2 \} \]

\[ B \oplus (A \otimes P) = \{ B_0 \oplus (A_0 \otimes P_0), B_1 \oplus (A_1 \otimes P_0), B_0 \oplus (A_0 \otimes P_1), B_1 \oplus (A_1 \otimes P_1), B_0 \oplus (A_0 \otimes P_2), B_1 \oplus (A_1 \otimes P_2) \} \]

\[ B \oplus_{\text{Taverna}} (A \otimes P) = \{ B_0 \oplus (A_0 \otimes P_0), B_1 \oplus (A_1 \otimes P_0) \} \]
One-to-one composition makes sense if data sets are correlated

$$\mathbf{B} \oplus (\mathbf{A} \otimes \mathbf{P}) = \left\{ \begin{array}{l}
B_0 \oplus (A_0 \otimes P_0), \\
B_0 \oplus (A_0 \otimes P_1), \\
B_0 \oplus (A_0 \otimes P_2), \\
B_1 \oplus (A_1 \otimes P_0), \\
B_1 \oplus (A_1 \otimes P_1), \\
B_1 \oplus (A_1 \otimes P_2) \end{array} \right\}$$

if A and B are correlated (application dependent, user defined)

- Sequential order may be relevant (but not reliable in case of a data- and service-parallel workflow enactor)
- Our hypothesis: explicit description of correlated data sets, or sequential order (default behavior)

No unique answer: depends on application expressiveness needs
Explicit correlation through groups

- The user defines correlation groups:
  - \( G = \{ (A_0, B_0), (A_1, B_1), \ldots \} \)
  - No relation between \( A_i \) and \( P_k \)

- Service \( S_1 \):
  - \( \oplus \) composition: \( A_i \) and \( B_j \) combined iff \( i=j \)

- Service \( S_4 \):
  - \( (A_i \oplus B_i) \otimes P_k \) and \( (A_j \oplus B_j) \otimes Q_m \) combined iff \( i=j \)
  - \( ((A_i \oplus B_i) \otimes P_k) \oplus ((A_i \oplus B_i) \otimes Q_m) \) for all \( k \) and \( m \)

\[ ((A \oplus B) \otimes P) \oplus ((A \oplus B) \otimes Q) \]
Algorithm

- Implement the semantics for any workflow graph
1. Build the workflow directed graph
1. Build the workflow directed graph
2. Add data groups to this graph
1. **Build the workflow directed graph**
2. **Add data groups to this graph**
3. **Initialize the directed acyclic data graph**
   1. Create root nodes from groups
1. **Build the workflow directed graph**
2. **Add data groups to this graph**
3. **Initialize the directed acyclic data graph**
   1. Create root nodes from groups
   2. Root nodes for orphan data
Algorithm

1. **Build the workflow directed graph**
2. **Add data groups to this graph**
3. **Initialize the directed acyclic data graph**
   1. Create root nodes from groups
   2. Root nodes for orphan data
4. **Start workflow execution**
1. **Build the workflow directed graph**
2. **Add data groups to this graph**
3. **Initialize the directed acyclic data graph**
   1. Create root nodes from groups
   2. Root nodes for orphan data
4. **Start workflow execution**
5. **At each service invocation**
   1. Update data graph
1. Build the workflow directed graph
2. Add data groups to this graph
3. Initialize the directed acyclic data graph
   1. Create root nodes from groups
   2. Root nodes for orphan data
4. Start workflow execution
5. At each service invocation
   1. Update data graph
1. **Build the workflow directed graph**

2. **Add data groups to this graph**

3. **Initialize the directed acyclic data graph**
   1. Create root nodes from groups
   2. Root nodes for orphan data

4. **Start workflow execution**

5. **At each service invocation**
   1. Update data graph
   2. Loop until no more data available
Implicit grouping of orphan input data sets composed by a one-to-one operator
Data fragments

- A service may produce more (or less) data than it consumes

\[ A = \{A_0\} \]

\[ B = \{B_0, B_1, B_2\} \]

- But this breaks the data parallelism assumption!
Bronze Standard application example

AGIR

Data composition pattern in service-based workflows, J. Montagnat et al, WORKS'06

www.aci-agir.org
Conclusions

- **Service-based workflow enactors**
  - User friendly approach
  - Well suited for scientific, data-intensive applications

- **Data composition patterns**
  - Very compact framework
  - Powerful expressiveness
  - Non-trivial operators semantics

- **Perspectives**
  - Data parallelism with data fragments
  - More composition patterns (all-to-all-but-one...)
  - Different semantics for one-to-one composition (one-to-one-inclusive, one-to-one-strict...)
• MOTEUR code and tutorial
  – http://www.i3s.unice.fr/~glatard

• Publications
  – Overview: Tristan Glatard et al, I3S tech report #06-07, HPDC'06
    http://www.i3s.unice.fr/~7Emh/RR/2006/RR-06.07-T.GLATARD.pdf
  – Overview, performances: Tristan Glatard et al. HPDC'06 poster
    http://www.i3s.unice.fr/~johan/publis/HPDC06.pdf
  – Software architecture: Tristan Glatard et al. GELA'06 (HPDC)
    http://www.i3s.unice.fr/~johan/publis/GELA06.pdf
  – Performances: Tristan Glatard et al. EXPGRID'06 (HPDC)
    http://www.i3s.unice.fr/~johan/publis/EXPGRID06.pdf
  – Medical imaging: Tristan Glatard et al. HealthGrid'06
    http://www.i3s.unice.fr/~johan/publis/HealthGrid06b.pdf
Shortcomings of the task-based approach

• The task-based approach mixes processing description and target data:
  – Static description of tasks
  – Usually single execution per Job Description File
    • Why are multiple-data jobs submitter so rare?
  – Tedious invocation process: first write a Job Description File

• Every piece of data is a file
  – Specifying input parameters (int, string, ...) to a job is not possible

• But legacy code execution is straight-forward
  – Just write the command line
• Only acyclic graphs are possible in the task-based approach
• Description is static
• Example: optimization loop could not be described
Data synchronization are difficult to describe

- Example: computing an average input

\[
A_0, A_1, A_2, \ldots \quad \text{\&} \quad B_0, B_1, B_2, \ldots
\]

\[
\begin{align*}
\text{Service0} & \quad \text{Service1} \\
\text{Service2} & \quad \text{\&}
\end{align*}
\]

\[
\text{Mean} (S_0(A_0), S_0(A_1), S_0(A_2), \ldots \quad S_1(B_0), S_1(B_1), S_1(B_2), \ldots)
\]
**Sequential order semantic**

A = \{A_0, A_1\}

B = \{B_0, B_1\}

C = \{C_0, C_1, C_2\}

\[ \begin{align*}
A \oplus_{\text{Taverna}} (B \otimes C) &= \{ A_0 \oplus (B_0 \otimes C_0), \ A_1 \oplus (B_1 \otimes C_0) \} \\
(A \oplus_{\text{Taverna}} B) \otimes C &= \{ \forall i, (A_0 \oplus B_0) \otimes C_i, \ \forall i, (A_1 \oplus B_1) \otimes C_i \} \\
B \oplus_{\text{Taverna}} (A \otimes P) &= \{ B_0 \oplus (A_0 \otimes P_0), \ B_1 \oplus (A_1 \otimes P_0) \}
\end{align*} \]