Taskflow-oriented Programming: A Method to Configure a Universal Client for Distributed Collaborative Computing

Franc Brglez, Hemang Lavana
Collaborative Benchmarking Laboratory (CBL)
Dept. Comp. Science, NC State University
http://www.cbl.ncsu.edu/

Contents

Background and Motivation
Related Work
Project Drivers and GUI
Task Architecture and XML Schema Representation
A Demo with Two Simple Tasks
A Demo with Concurrent Executions

TaskFlow Architecture
Collaborative Computing with Taskflows
Conclusion and Summary
Background and Motivation

Megaprogramming (Wiederhold 1992)
Interactive programming (Wegner 1997)

What problem are we solving?
What is a whitebox?
Simple examples to illustrate needs
What about protocols

What Problem are We Solving?

Given: L blackbox components (executable programs) and test data,
M whitebox components (explained below),
N owners (up to L+M),
K hosts (up to L+M).

Problem: create a new whitebox component and execute

Definitions:

Task is a process of invoking and executing a component
Taskflow is a directed graph with tasks and data as vertices,
task-to-task control-edges and data-to-task or task-to-data data edges
Whitebox component is a taskflow containing up to L+M components
What is a Whitebox Component?

**White box component** is a taskflow containing up to L+M components:

**Programmable**
- tasks can be sequenced serially or concurrently
- the order of sequence can be decided by data
- task may be repeated a number of times
- only one entry and one exit point

**Interactive, user-configurable during runtime**
- enabling/disabling of control edges to control the scope of execution
- enabling/disabling taskflow for execution with local or global flow data

**Collaborative**
- asynchronous mode
- synchronous mode

**Platform-independent**
- accessible from a web browser

Simple Examples to Illustrate Needs

**Task chaining** (download address and print)

**Task concurrency** (check airfares with two on-line agents)

**Task synchronization** (select best fare and reserve itinerary)

**Task cancellation** (cancel 3rd query if reasonable price with first two)

**Task reconfiguration** (add one more on-line agent for search)

**Synchronous collaboration** (ask spouse to confirm itinerary)

**Asynchronous collaboration** (notify company to pay the fare)

**Platform-independent** (web-based access)
What Protocols Can We Use

... when creating a whitebox?

If components are X-compliant, where \( X = \{\text{rpc, CORBA, JavaRMI, SOAP}\} \), protocols such as rpc, CORBA, JavaRMI, SOAP may be used.

If components are not X-compliant, protocols such as ftp, telnet, http, ssh may be used.

In this research, we deal with components that are not X-compliant.

Related Work

- ToolWire
  http://www.toolwire.com
- Web-Based Electronic Design (WELD), WebTop, and CollabTop
  http://www-cad.EECS.Berkeley.EDU/Respep/Research/weld
  http://www-mit.mit.edu/CAPAM/demo-posters/deba
- Groupkit
  http://www.cpsc.ucalgary.ca/grouplab/groupkit
- Collaboratory for the Web: TANGO
  http://trurl.npac.syr.edu/tango
- VNC: Virtual Network Computing
  http://www.uk.research.att.com/vnc
Highlights of on-going research

• a taskflow architecture to support concurrency
• a highly interactive GUI-based taskflow design and execution environment
• support for task ownership during asynchronous collaboration
• rendering collaboration-unaware tcl-applications collaborative
• web-accessibility of standalone tcl-applications

Ranking Programming Languages for Concurrency Support

System languages (Ada, Java, ...)
A taskflow environment (cntML, based on XML)
Scripting languages (perl, python, tcl, ...)

The taskflow environment extends the capabilities of scripting languages twofold:
(1) concurrency support without the complexity of thread programming
(2) a highly interactive GUI taskflow design and execution environment
Project Drivers and GUI

OpenDesign project (year 2000 of DARPA-funded Vela project)
A prototype infrastructure for distributed and collaborative VLSI systems design, bringing together a number of university-based and commercial electronic design automation tools.

OpenExperiment project (year 2000 of DARPA-funded Vela project)
A prototype infrastructure for distributed and collaborative experimental design, evaluating the performance of algorithms solving NP-hard problems, including the generation of equivalence class data sets.

OpenWriter project (senior class project)
A prototype infrastructure for distributed and collaborative writing, editing, and camera-ready technical report, article, and book composition, using a number of open source software and utilities.

Updates
http://www.ncsu.edu/OpenProjects/
A Tree View of a Hierarchical Taskflow

- (taskFlowInstance1
  - InputList
    - mIn1 = “fileIn1.t”
    - mIn2 = “fileIn2.t”
  - OutputList
    - mOut1 = “fileOut1.r”
  - TaskList
    - (Begin) |x|
    - (TaskA) |o| |x|
      + InputList
      + OutputList
      + TaskList
    - (TaskB) |x|
      + InputList
      + OutputList
    + (TaskC) |o| |x|
    - (End)

invoke task
open graphView

multi task
single task
A Graph View of a Hierarchical Taskflow

Execute task by clicking the “task button”
Abort task by clicking the “flashing task button”

FSMD states:
- white (task not yet executed)
- yellow (task “aborted”)
- green (task “executing”)
- blue (task forkCondition is “valid”)
- violet (task forkCondition is “invalid”)
- pink (task “timed-out”)
- red (task “aborted”)

Task Architecture and XML Schema Representation

EncapsulatedSingleTask (FSMD)
- SingleTaskDefn (STD)
  - InputPortList
  - InOutPortList
  - OutInPortList
  - OutputPortList
- SingleTaskBody
  - BeginFork
  - BlackBox Comp.
  - EndJoin

EncapsulatedMultiTask (FSMD)
- MultiTaskDefn (MTD)
  - InputPortList
  - InOutPortList
  - OutInPortList
  - OutputPortList
  - TaskInstanceList
  - TaskGraph
- MultiTaskBody
  - BeginFork
  - TaskInstance1
  - ... EndJoin
  - DataGraph

TaskInstance
- ControlJoin
- DataMux
- EncapsulatedTask
- ControlFork

MainTask
- TaskInstanceList
- TaskGraph
- BeginFork
- TaskInstance1
- ... EndJoin
XML Schema Specification for MTD

<!-- type declaration of multi task definition -->
<xsd:complexType name="MultiTaskDefn" content="elementOnly">
  <xsd:sequence minOccurs="0" maxOccurs="1">
    <xsd:element name="Title" type="Title"/>
    <xsd:element name="Description" type="Description"/>
    <xsd:element name="InputList" type="InputList"/>
    <xsd:element name="InoutList" type="InoutList"/>
    <xsd:element name="OutputList" type="OutputList"/>
    <xsd:element name="TaskList" type="TaskList"/>
    <xsd:element name="ControlGraph" type="textOnly"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

XML Editor for Taskflow Construction

An XMLspy editor validates taskflow during construction itself
Y_taskFlowSchedulingEngine

XML Taskflows

Tcl-xml Parser

Validate Taskflow Specs

Display Taskflow Tree, Graph

Taskflow Scheduler

Taskflow Scheduler

Begin

Evaluate ControlJoin

False

Noop

Enable Task

Evaluate ControlFork

False

Noop

Execute BBC or expand WBC

Foreach child task

No more tasks

End

A Demo with Two Simple Tasks (single pSolver)

Descriptions

- singleParabolaTask-demo
- O0InvokeMe
- initData
- parabola_body.ctd
- parabola_defn.ctd
- parabola_main.ctd
- pSolver.tcl

Demos

- hierarchical task tree/graph views
- taskflow simulation execution
- taskflow local data execution
- edit data represented by data I/O nodes
- taskflow flow data execution

BlackBox
parabola_main.ctd

<MainTask>
  <TaskList>
    <Begin/>
    <Task instance="(parabola)" \
      taskRef="parabola_defn.ctd#task_parabola"/>
    <End/>
  </TaskList>
  <TaskGraph>
    (BEGIN) => (parabola) => (END)
  </TaskGraph>
  <TaskInstance instance="(parabola)">
    <SetInput port="mInDescrA1">
      <LocalValue> pbInDescrA1.d </LocalValue>
    </SetInput>
    <SetInput port="mInitCost">
      <LocalValue> 1e6 </LocalValue>
    </SetInput>
    <SetOutput port="mOutReport">
      <LocalValue> parabola.rpt </LocalValue>
    </SetOutput>
  </TaskInstance>
</MainTask>

parabola_defn.ctd

<MultiTaskDefn name="task_parabola" \
  bodyRef="parabola_body.ctd#task_parabola">
  <InputList>
    <Input port="mInDescrA1" type="persistent">
      <Title> Input description for task A </Title>
    </Input>
    <Input port="mInitCost" type="temporary">
      <Title> Initial cost (should be very large) </Title>
    </Input>
  </InputList>
  <OutputList>
    <Output port="mOutReport" type="persistent">
      <Title> Output report of parabola cost function evaluation </Title>
    </Output>
  </OutputList>
  <TaskList>
    <Begin/>
    <Task instance="(InitA1)" taskRef="parabola_defn.ctd#pInitSolver"/>
    <Task instance="(A1)" taskRef="parabola_defn.ctd#pSolver"/>
    <End/>
  </TaskList>
  <TaskGraph>
    (BEGIN) => (InitA1) => (A1) => (END)
    (A1) => (A1)
  </TaskGraph>
</MultiTaskDefn>

Note the backedge (self-loop)
parabola_body.ctd (1)

<MultiTaskBody name="task_parabola" sleep=""
<BeginFork/>

<TaskInstance instance="(A1)" maxIterate="10" host="h1">
<DataMux>
<SetInput port="inpDescr">
<LocalValue> taskA1_inpDescr.d </LocalValue>
</SetInput>
<SetInput port="initCost">
<LocalValue> 1.e6 </LocalValue>
</SetInput>
<SetInput port="initSoln">
<LocalValue> 5 </LocalValue>
</SetInput>
<SetOutIn port="newCost">
<LocalValue> taskA1_newCost.d </LocalValue>
</SetOutIn>
<SetOutIn port="newSoln">
<LocalValue> taskA1_newSoln.d </LocalValue>
</SetOutIn>
<SetOutPut port="nIter">
<LocalValue> taskA1_nIter.d </LocalValue>
</SetOutPut>
</DataMux>
<RepeatCondition>
<UserRepeat>
evalRepeatCondition \\
"[cdtReadData newCost]" IsLessThan "[cdtReadData oldCost]"
</UserRepeat>
</RepeatCondition>
</TaskInstance>

parabola_body.ctd (2)

<SetOutPut port="oldCost">
<LocalValue> taskA_oldCost.d </LocalValue>
</SetOutPut>
</DataMux>
<RepeatCondition>
<UserRepeat>
</UserRepeat>
</RepeatCondition>
</TaskInstance>

<SingleTaskBody name="pSolver" sleep=""
<ExecCommand type="TclCmd">
<Value>
pSolver.tcl "[cdtGetData inpDescr]" "[cdtGetData initCost]" \ 
"[cdtGetData initSoln]" "[cdtGetData newSoln]" \ 
"[cdtGetData newCost]" "[cdtGetData nIter]" \ 
"[cdtGetData oldCost]"
</Value>
</ExecCommand>
</SingleTaskBody>
Single Parabola Demo Outline

-- introduce hierarchical task tree and graph views
    turn off/on control edges

-- introduce taskflow simulation mode
    enter 1 second wait/task (random variable otherwise. 0--5 seconds)
    execute taskflow with all edges turned on

-- introduce taskflow local data execution
    probe data nodes for data values expected by task (initA1)
    probe data I/O nodes for data values assigned to task (A1)
    (not all data is present yet)
    turn off all edges except (initA1) -> (A1)
    click on (initA1), continue clicking on (A1) until “minimum” is declared
    probe data I/O nodes for data values assigned to task (A1)

-- edit data represented by data I/O nodes
    click on pbInDescrA1.d an edit contents (but don’t) save

-- introduce taskflow flow data execution (here, we get the same results)
    make sure all edges are connected, reset and clean all data
    click on (Begin)
    probe data I/O nodes for data values assigned to task (A1)

A Demo with Concurrent Task Executions

see the 2-out-3 synchronization pattern in ..._body.cdt (next page)

A1, A2, A3 should execute concurrently if on different hosts
<MainTask>
  <TaskList>
    <Begin/>
    <Task instance="(parabola)" taskRef="parabola_defn.ctd#task_parabola"/>
    <End/>
  </TaskList>
  <TaskGraph>
    (BEGIN) => (parabola) => (END)
  </TaskGraph>

  <TaskInstance instance="(parabola)">
    <SetInput port="mInDescrA1">
      <LocalValue> pbInDescrA1.d </LocalValue>
    </SetInput>
    <SetInput port="mInDescrA2">
      <LocalValue> pbInDescrA2.d </LocalValue>
    </SetInput>
    <SetInput port="mInDescrA3">
      <LocalValue> pbInDescrA3.d </LocalValue>
    </SetInput>
    <SetInput port="mInitCost">
      <LocalValue> 1e6 </LocalValue>
    </SetInput>
    <SetOutput port="mOutReport">
      <LocalValue> parabola.rpt </LocalValue>
    </SetOutput>
  </TaskInstance>
</MainTask>

......_defn.ctd

<TaskList>
  <Begin/>
  <Task instance="(InitA1)" taskRef="parabola_defn.ctd#pInitSolver"/>
  <Task instance="(InitA2)" taskRef="parabola_defn.ctd#pInitSolver"/>
  <Task instance="(InitA3)" taskRef="parabola_defn.ctd#pInitSolver"/>
  <Task instance="(A1)" taskRef="parabola_defn.ctd#pSolver"/>
  <Task instance="(A2)" taskRef="parabola_defn.ctd#pSolver"/>
  <Task instance="(A3)" taskRef="parabola_defn.ctd#pSolver"/>
  <Task instance="(B)" taskRef="parabola_defn.ctd#pEvaluator"/>
  <Task instance="(C)" taskRef="parabola_defn.ctd#pReport"/>
  <End/>
</TaskList>
<TaskGraph>
  (BEGIN) => (InitA1) => (A1) => (B) => (C) => (END)
  (A1) => (A1)
  (BEGIN) => (InitA2) => (A2) => (B) => (C) => (END)
  (A2) => (A2)
  (BEGIN) => (InitA3) => (A3) => (B) => (C) => (END)
  (A3) => (A3)
</TaskGraph>
</MultiTaskDefn>
Concurrent Executions Demo Outline

-- taskflow simulation mode
click “exec checkboxes” so that tasks B and C are always executed ...
do task (A1) by repetitive clicks (with repeat-edge off)
do task (A3) single node with repeat-edge on
when (A3) is completed task (B) should fire (regardless of (A2) state)
enter 2 seconds of sleep/task, do the entire taskflow, blocking one edge
(watch the concurrent 3-task execution)
modify source file to have tasks A1, A2, A3 executed on single host
(forcing a serial execution of the three tasks)

-- taskflow local data execution
turn-off all edges to (B)
check for input data and lack of output data
do tasks (A1), (A2), and (A3) concurrently
check that now we have input data for (B) but not yet (C)
click on (B) which invokes (C)

-- taskflow flow data execution
connect the entire flow
check for input data and lack of output data
do tasks (A1), (A2), and (A3) concurrently
check that now we have input data for (B) but not yet (C)
click on (B) which invokes (C)
connect the entire flow and re-execute to get final results in a single click ....
TaskFlow Architecture

- Task Primitives
- Task Primitives Encapsulation
- Task Primitives Instances
- TaskFlow Graph Model
- Two State One Task Example (in hardware)
- Four State One Task Example (in hardware)
- Task Architecture
- Task FSMD
- TaskFlow Views
  - TaskFlow Scheduling Engine

Y_taskPrimitives-a

- control port for task invocation (input)
- control port for task execution status (output)
- control port for task abort (input/output)
- port for temporary/persistent data (input/output/inout)
- port for temporary/persistent data pair (input/output/inout)
- port for temporary/persistent data stack (input/output/inout)
- port for temporary/persistent data pair stack (input/output/inout)
- temporary/persistent data
- temporary/persistent data stack
- control abort connector (port-port, opened/closed by user)
- control invocation connector (port-port, opened/closed by user)
- control invocation connector (port-port, always closed)
- data connector (data-port, port-data or port-port)
Y_taskPrimitives-b

From predecessor tasks

To successor tasks

flowData/localData

reset

DFM

BBC data

FSMD

task config. data

BBC data

Black-Box component

Encapsulated BBC

Encapsulated WBC

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset

reset
Y_taskPrimitivesEncap-b

WBC (White-Box Component)

TaskInstance1

TaskInstance2

TaskInstance3

Y_taskPrimitivesInstance-c

Task Instance

Encap. BBC/WBC

FSMD

task config. data

BBC or WBC data

user

To

successor

tasks

CJ

Reset

user

From

predecessor

tasks

flowData/localData

DM

Din (flow)

Din (local)

Dout

CF
For each user invocation, for each user abort edge, there is a corresponding program invocation and program abort edge.
**Y_taskArchitecture**

From M predecessor tasks

- \( C_{li} \)
- \( E_{fi} \)
- \( F_i \)
- \( L_i \)

flowData/localData

Invoke Abort Skip

Reset

Task config. data

FSMD

Waiting

Enabled

executing

completed

Done

Aborted

TimedOut

Skipped

To N successor tasks

- \( C_{Oi} \)

**Y_taskFSMD**

Finite State Machine

(as per FSM transition table)

\( \{ P_i, P_a, P_s \} \)

\( R \)

Datapath

(as per datapath table)

\( \{ Q_E, Q_X, \ldots, Q_S \} \)

invocation signal

completion status

BBC or WBC

D_in

D_out
Y_taskFlowViews1-a

- (taskFlowInstance1
  - InputList
    • mIn1 = “fileIn1.t”
    • mIn2 = “fileIn2.t”
  - OutputList
    • mOut1 = “fileOut1.r”
  - TaskList
    • (Begin) |x|
    - (TaskA) |o|x|
      + InputList
      + OutputList
      + TaskList
    - (TaskB) |x|
      + InputList
      + OutputList
    + (TaskC) |o|x|
    • (End)

- (taskFlowInstance2
  - InputList
    • mIn1 = “fileIn1.t”
  - OutputList
    • mOut1 = “fileOut1.r”
  - TaskList
    • (Begin) |x|
    - (TaskA) |o|x|
      + InputList
      + OutputList
      + TaskList
    - (TaskB) |x|
      + InputList
      + OutputList
    + (TaskC) |o|x|
    • (End)

Y_taskFlowViews1-b

hierarchical task
(TaskA)  Execute task by clicking the “task button”
(TaskB)  Abort task by clicking the “flashing task button”
(TaskC)
mIn1 = “fileIn1.t”
token holder states:
• white  (task not yet executed)
• yellow  (task “aborted”)
• green   (task “executing”)
• blue    (task forkCondition is “valid”)
• violet  (task forkCondition is “invalid”)
• pink    (task “timed-out”)
• red     (task “aborted”)

user-controlled edge
primitive task
Collaborative Computing with Taskflows

- Asynchronous Collaboration
- Synchronous Collaboration
- Asynchronous and Synchronous Collaboration
- Web-Interface to Taskflow Environment
Asynchronous Collaboration: Task and Data Ownership

Ownership assignment to tasks
- Input and output data inherit ownership of task

Shared data at ownership boundaries (conflict)
- Use read/write locks to resolve conflicts
- Add `buffer` task with dual ownership

Two users U1 and U2
- U1(A), U1(B)
- U2(C), U2(D)

Asynchronous Collaboration: Ownership Locking

Asynchronous Group Server (AGS)
Asynchronous Collaboration: Buffer Task

Synchronous Collaboration: GroupKit Vs VNC
Synchronous Collaboration Architecture

Synchronous Group Client (SGC)

Synchronous Group Server (SGS)

Selective Display Synchronization

Asynchronous and Synchronous Collaboration: OmniDesk

Asynchronous and Synchronous Collaboration: OmniDesk

OmniDesk (AGS+SGS)

Mutual Consent

OmniFlow + SGC

OmniFlow + SGC
**Web-Interface for the Environment**

Web-browsers are here to stay (for a while anyway ...)

Questions:

- Can we use a web-browser for our environment?

Features expected (at least):

- hierarchy of windows
- no tedious installation
- access to file system, printing
- security of applets

---

**Tcl-Plugin: Restrictions and Solutions**

Restrictions of Tcl-plugin code:

- no auto_load mechanism
- partial set of Tk widgets
- no standard input/output

Current coding techniques for Tcl-plugins:

- merge all tcl application scripts into a single file
- require each client to install the application package
- (re-)design the application GUI to contain a single window
- use an appropriate security policy
WebWiseTclTk Architecture

Downloads the toolkit as well as the tclet scripts from the web server

Implementation of a ‘toplevel’ Window Using Frames

Resize window in both directions simultaneously
Resize window in vertical direction
Kill
Window Title
Iconify
Maximize
Menubar
Frame corresponding to toplevel window
Resize window in vertical direction
Resize window in horizontal direction
Example: TkWidget Demos

Evaluation of WebWiseTclTk using TkWidget Demos

<table>
<thead>
<tr>
<th>Category</th>
<th>Scripts</th>
<th>Action events</th>
<th>Pass</th>
<th>Fail</th>
<th>Diag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buttons, ...</td>
<td>8</td>
<td>(1+4+3+5+2+4+1+3)</td>
<td>22</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Listboxes</td>
<td>3</td>
<td>(3+4+3)</td>
<td>9</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Entries</td>
<td>3</td>
<td>(6+7+2)</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>5</td>
<td>(3+3+2+6+2)</td>
<td>15</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Canvases</td>
<td>7</td>
<td>(4+2+5+3+2+2+2)</td>
<td>17</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Scales</td>
<td>2</td>
<td>(1+1)</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Menus</td>
<td>2</td>
<td>(5+6)</td>
<td>9</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Dialogs</td>
<td>3</td>
<td>(1+2+1)</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>3</td>
<td>(1+1+1)</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>36</strong></td>
<td><strong>104</strong></td>
<td><strong>91</strong></td>
<td><strong>5</strong></td>
<td><strong>8</strong></td>
</tr>
</tbody>
</table>
Conclusion and Summary

Key contributions of this work:

• a taskflow architecture to support concurrency
• a highly interactive GUI-based taskflow design and execution environment
• support for task ownership during asynchronous collaboration
• rendering collaboration-unaware tcl-applications collaborative
• web-accessibility of standalone tcl-applications

Benefits:

• scalability: large user-configurable programs can be composed from existing legacy programs for network-transparent cross-platform execution.
• flexibility to control standalone execution of subtasks.
• flexibility to decompose any computing project into a hierarchy of tasks,
  -- bind the tasks and data to taskflow instances,
  -- delegate the subtasks to distributed team members,
  -- support asynchronous and synchronized execution policies.

Future work:

• extended formalization of the control/data taskflow model.
• optimize resource utilization and perform load balancing.