Automatic workflow construction

Viktoras Veitas
vveitas@gmail.com

May 9, 2016

Introduction

We want to specify a system which is able to automatically construct workflows from the independent components, which is a general coordination problem, as defined by Heylighen, 2013 (see Figure 1).

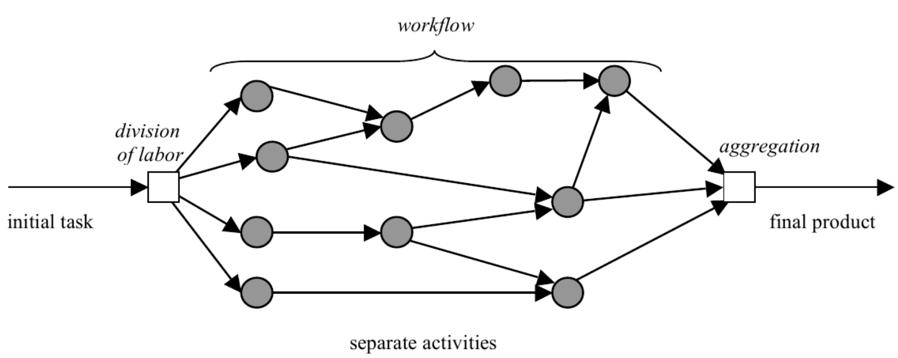


Figure 1: Coordination in which an initial task is split up in separate activities performed by different agents (division of labor), which are followed by other activities (workflow), and whose results are assembled into a final product (aggregation). Grey circles represent individual agents performing activities. Arrows represent the "flow" of work from one agent to the next.

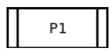
System specification

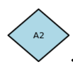
A single most important prerequisite for the algorithm to work is that the graph shoud be connected - i.e. each vertex should be accessible from some vertex in the graph.

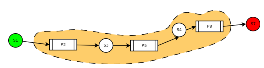
Objects

Informal specification

- **Data holder**  is non-ambiguously describable state of the data / objects. In offer network framework, simply speaking, data holders are descriptions of offers and demands which have been entered into the system.

- **Work**  is best understood as a connector between demands and offers. It is convenient to describe work within the workflow framework as a process, which transforms demand (which can be potentially satisfied by the network) into supply (which could be potentially taken by the network). Work is equivalent to the *reaction* in Reaction Networks / Chemical Organization Theory frameworks Heylighen et al., 2015. states.

- **Agent** . Every work has to be neccessarily 'owned' by an agent. An agent can own more than one work. Agents allow for the 'real world users (persons, programs)' to interface with the offer network.

- **Chain**  is a collection of works connected via their demands and offers. I.e. suppose the network contains a work $P1$ which demands $D1$ and offers $D2$ and a work $P2$ which demands $D3$ and offers $D4$. Further suppose that $D1 == D2$. In this case $P1$ and $P3$ can be connected into a chain which, taken as a whole demands $D1$ and offers $D4$. If offer network is represented as a graph structure, a chain is equivalent to the legal path in the graph.

- **Loop** is simply a closure of a chain. It is equivalent to the *closed walk* or a *cycle* in terms of graph theory.

Initially the offer network consists of the number of unconnected works with their demands, offers and, optionally, agents which own these works (see Figure 2.

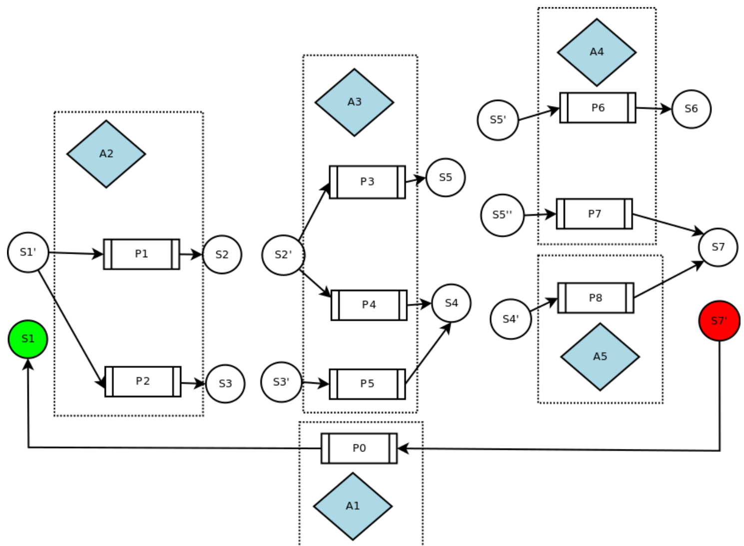


Figure 2

The goal of the system is to find loops by finding matching {*demand*, *offer*} pairs and connecting them (see Figure 3).

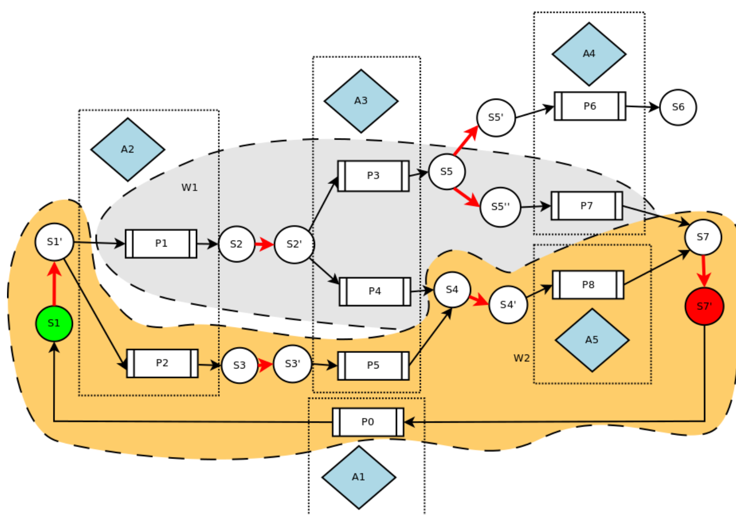


Figure 3: Data centric approach, describing workflow as a sequence of state transitions (inspired by Cushing, 2015; Cushing et al., 2015).

Semi-formal specification

- The structure is a list of vertexes and edges:

$$N = \{V, E\} \tag{1}$$

- Vertexes can be of type 'agent', 'work' or 'item':

$$\begin{aligned}\forall V(&type(V, \text{agent})\\ &\vee type(V, \text{work})\\ &\vee type(V, \text{item}))\end{aligned} \tag{2}$$

- Vertexes can be connected with edges, which define predicate relations between them:

$$\begin{aligned}&\exists V1 \exists V2 \exists E(connected(V1, E, V2)\\ &\qquad \Leftrightarrow \exists pred(pred(V1, V2) \wedge type(E, \text{pred})))\end{aligned} \tag{3}$$

- Edges can be of type 'owns', 'offers', 'demands' and 'similar' (which correspond to the possible predicate relations between vertexes):

$$\begin{aligned}\forall E(&type(E, \text{owns})\\ &\vee type(E, \text{similar})\\ &\vee type(E, \text{offers})\\ &\vee type(E, \text{demands}))\end{aligned} \tag{4}$$

- Agents 'own' works:

$$\begin{aligned}owns(V1, V2) :\Leftrightarrow &\forall V1 \forall V2 \forall E(type(V1, \text{agent}) \wedge type(V2, \text{work})\\ &\wedge connected(V1, E, V2) \wedge type(E, \text{owns}))\end{aligned} \tag{5}$$

- Works 'demand' and/or 'offer' data :

$$\begin{aligned}demands(V1, V2) :\Leftrightarrow &\forall V1 \forall V2 \forall E(type(V1, \text{work}) \wedge type(V2, \text{data})\\ &\wedge connected(V1, E, V2) \wedge type(E, \text{demands}))\end{aligned} \tag{6}$$

$$\begin{aligned}offers(V1, V2) :\Leftrightarrow &\forall V1 \forall V2 \forall E(type(V1, \text{work}) \wedge type(V2, \text{data})\\ &\wedge connected(V1, E, V2) \wedge type(E, \text{offers}))\end{aligned} \tag{7}$$

$$\forall V1(type(V1, \text{work})\, \exists V2(demands(V1, V2) \vee offers(V1, V2))) \tag{8}$$

- Data holders can be connected with similarity relation wich has a value between 0 and 1:

$$\begin{aligned}similar(V1, V2) :\Leftrightarrow &\forall V1 \forall V2 \forall E(type(V1, \text{data}) \wedge type(V2, \text{data})\\ &\wedge connected(V1, E, V2) \wedge type(E, \text{similar}))\end{aligned} \tag{9}$$

$$\forall E(type(E, \text{similar}) \Rightarrow \exists! value(E, \text{s})); s : \mathbb{R}[0, 1] \tag{10}$$

- If values of data vertexes are fixed-lenght binary strings, the value of similarity relation is defined by cosine similarity:

$$\forall V(type(V, \text{data}) \Rightarrow value(V, \text{bs})); bs : \text{binary string of lenght } n \tag{11}$$

$$\begin{aligned} similar(V1, V2) :\Leftrightarrow & \exists! E \exists! s(value(E, \text{s}) \wedge \\ & \exists! bs1 \exists! bs2(value(V1, \text{bs1}) \wedge value(V2, \text{bs2}) \\ & \wedge s = \frac{bs1 \cdot bs2}{||bs1|| \cdot ||bs2||}) \end{aligned} \tag{12}$$

Processes

The goal of the algorithm is to find loops of {*offer*, *demand*} pairs by traveling via their chains.

Informal specification

2.2.1.1 Basic processes

- **Connect similar**: from every *Data holder* vertex $i = 0$ in a graph, initiate the traversal of a fixed lenght *len* through neighbouring *Data holder* vertexes $j =$ and calculates the similarity for each pair of $\{i, j\}$. If similarity exceeds certain predefined threshold (e.g. data values are more 'similar' than 'dissimilar', which means cosine simillarity is greater than 0.5), then a link is created between these data holders and similarity measure recorded. This traversal effectively clusters the data holder nodes - therefore the path among similar data holders becomes very short.

- **Detect cycles**: All 'task' nodes start the traversal in parallel by find a path connecting its {*demand*, *offer*} pair. It follows the adapted algorithm from Rocha and Thatte, 2015. The difference is that the traversal records similarity measures of the all paths followed and therefore carries a measure of path cost. Furthermore, each 'task' node starts the traversal simulateneously.

- **Generate process**: A process can be generated only by agent which is a member of the network. A work is an 'atomic process', which is specified externally to the network. A task is also a 'compount process', which is made from the chain of 'atomic processes'. When an agent posts a task, the network helps it to find chains of atomic processes which 'closes the task' (see detect cyles);

- **Add agent**: A new agent can be added to the network only by the recommendation of the existing agent so that these agents get connected via 'knows' relation, which optionally could also carry trust value. This is needed because the network is decentralized, therefore there is no 'central' authority which could regulate who can / cannot join the network. Also, this scheme ensures that the all the all the processes are connected, which is the single most important prerequisite for the system to work.

Semi-formal specification

- **Connect similar**

```
operation START() is
  for each id_n ∈ neighbors_i do

}
```

Figure 4: Finding matches process

- **Detect cycles**:

```
function COMPUTE(Mi)
 if i = 0 then
 for each w in N +(v) do
  send (v) to w
 else if Mi = empty then
  deactivate v and halt
 else
  for each (v1 , v2, . . . , vk ) in Mi do
   if v1 = v and min{v1, v2 , . . . , vk } = v then
    report (v1 = v, v2 , . . . , vk , vk+1 = v)
   else if v in {v2 , . . . , vk } then
    for each w in N + (v) do
     send (v1 , v2 , . . . , vk , v) to w
```

Figure 5: 'Detect cycles' process listing

References

Cushing, R. S. (2015). *Data-centric computing on distributed resources.*

Cushing, Reginald et al. (2015). "Towards a data processing plane: An automata-based distributed dynamic data processing model". In: *Future Generation Computer Systems.*

Heylighen, Francis (2013). "Self-organization in Communicating Groups: The Emergence of Coordination, Shared References and Collective Intelligence". en. In: *Complexity Perspectives on Language, Communication and Society.* Ed. by Àngels Massip-Bonet and Albert Bastardas-Boada. Understanding Complex Systems. Springer Berlin Heidelberg, pp. 117–149.

Heylighen, Francis et al. (2015). *Chemical Organization Theory as a modeling framework for self-organization, autopoiesis and resilience.* GBI Working Paper v1, p. 29.

Rocha, Rodrigo Caetano and Bhalchandra Digambar Thatte (2015). “Distributed cycle detection in large-scale sparse graphs”. In: *XLVII SBPO 2015 - Simpósio Brasileiro de Pesquisa Operacional*.