Andrew Schumann, Krzysztof Pancerz, Andrzej Szelc

The Swarm Computing Approach to Business Intelligence

Studia Humana nr 15, 41-50

2015

DE GRUYTER OPEN

studia humana
QUARTERLY JOURNAL

Studia Humana
Volume 4:3 (2015), pp. 41—50
DOI: 10.1515/sh-2015-0019

The Swarm Computing Approach to Business Intelligence

Andrew Schumann

University of Information Technology and Management in Rzeszow,
Poland

e-mail: aschumann@wsiz.rzeszow.pl

Krzysztof Pancerz

University of Information Technology and Management in Rzeszow,
Poland

e-mail: kkpancerz@gmail.com

Andrzej Szelc

University of Information Technology and Management in Rzeszow,
Poland

e-mail: aszelc@wsiz.rzeszow.pl

Abstract:
We have proposed to use some features of swarm behaviours in modelling business processes. Due to these features we deal with a propagation of business processes in all accessible directions. This propagation is involved into our formalization instead of communicating sequential processes. As a result, we have constructed a business process diagram language based on the swarm behavior and an extension of that language in the form of reflexive management language.
Keywords: swarm intelligence, emergency, unconventional computing

1. Introduction

For designing the information systems of organisations the process algebras have been usually used to model networks of autonomous units communicating asynchronously via messages. These systems are

constructed as the service-oriented architecture. There are three types of business architectures: (i) static information systems where there are no structure changes during the system's runtime; (ii) dynamic information systems where there exist rules of system evolution; (iii) mobile information systems, where some components can change their context in the system's logical structure during its execution.

The formal language that is used for describing patterns of interaction in concurrent systems is represented by one of the forms of formalizations of *communicating sequential processes*. The first language of communicating sequential processes was introduced by Charles Hoare in 1978. These languages contain the items of two sorts: (i) communication events; (ii) processes which interact among themselves through message-passing communication. In these languages business processes can be coded algorithmically to simulate a real hierarchically composed business process ordered by some business rules leading to a business aim.

So, standardly, business processes are simulated by means of communicating sequential processes, i.e., in the form of sequences which can be defined algorithmically. There are many notations for representing real business processes within process algebras in communicating sequential forms: Business Process Modelling Notation, Business Process Execution Language for Web Services, Business Process Definition Metamodel, Event Driven Process Chain, Petri Nets, etc. [1], [4], [5], [6], [8], [10, [11], [13], [18]. These notations propose a business process diagram as an input transformation which is controlled by an interface. It is very useful to have a representation of real processes as a diagram where all entities are ordered algorithmically. However, in real organizations the requirements to define all tasks for the office staff as process algebra algorithms can complicate a real business process and overload the staff. Each staff member cannot be considered just automaton with some inputs and outputs.

In this paper, we propose how we can avoid sequentiality in modelling business processes. The matter is that communicating sequential processes reduce human resources to a kind of automata with inputs and outputs. But people can be tired or frustrated and sometimes it is better to have possibilities to define some tasks by the staff contextually. The contextuality of some tasks cannot be defined in the form of communicating sequential processes, because due to the contextuality we deal with an infinite set of atomic acts [14] and thus we cannot formalize business processes in the standard tools.

There is a kind of unconventional computing that is called *swarm computing* where we deal with processes which cannot be represented in the form of sequences. The point is that in swarms emergent effects appear usually, when the whole system cannot be considered an inductive composition of its subsystems. Hence, we can regard business organizations as swarms also, where there are some emergent effects, as well. In particular, due to these effects an organization can be more adaptive, when some roles are spontaneously redistributed in new changing contexts.

One of the swarms we have studied logically [2] is represented by the large one-cell organism of plasmodium of *Physarum polycephalum*. The plasmodium moves by protoplasmic streaming which reverses every 30-60s. It can switch its direction or even multiplies according to different biosignals attracting or repelling its behaviour. So, on the one hand, the plasmodium is a one-cell organism, but, on the other hand, it behaves as a swarm with splitting and multiplications.

Plasmodium motions can be controled by several instructions like: add node, remove node, add edge, remove edge [15], [16]. Adding and removing nodes can be implemented through activation and deactivation of attractants, respectively. Adding and removing edges can be implemented by means of repellents so that an activated repellent avoids a plasmodium transition between attractants. Hence, the plasmodium can be considered a programmable biological device in the form of a timed transition system, where attractants and repellents determine the set of all plasmodium transitions. As a result, in the case of plasmodium of *Physarum polycephalum*, we deal with a kind of process algebra. Its main

feature is that the plasmodium demonstrates a massive-parallel expansion in all possible directions. This feature is an emergent effect in plasmodium motions and cannot be represented in the linear form as sequences.

In *Physarum Chip Project: Growing Computers from Slime Mould* (PhyChip) [3] funded by the Seventh Framework Programme (FP7), we have constructed a biological computer on programmable behavior of *Physarum* plasmodium. We have designed a software tool for simulating the plasmodium motions. This tool, i.e., a programming language used for the plasmodium computer, can be used also for simulating business processes. In this simulation we can examine business organizations as swarms with a kind of emergency. In this paper we show how we can code communicating business processes in our language.

2. *Physarum* Business Process Diagram Language

A business process is a set of activities performed in an organization to realize a business goal. A business process language is designed to model the business processes. This language can be used to support the design, administration, and configuration of business processes. Our language for simulating the plasmodium behavior [12] can be reestablished as a business process diagram language. Let us call it the *Physarum business process diagram language* that will be used as a graphical notation for business process modeling, with an emphasis on swarm effects. In this language we well deal also with AND-split, AND-join, OR-split, OR-join, etc., but with a possible emergency in configurations.

Using the Java environment, we have constructed the software tool working under the client-server paradigm. Communication between clients and the server is realized through text messages. An example of code responsible for the creation of new events associated with stimuli for the plasmodium has the following form:

```
p1_a1=new Attractant(195,224,1);
p1_a2=new Attractant(541,310,1);
p1_a1=new Attractant(580,92,2);
p2_r1=new Repellent(452,130,2);
p2_r1=new Repellent(659,327,1);
```

The two first parameters of events constructors determine the location whereas the last parameter is the client ID. The initial client window for fixing different events is shown in Figure 1. Then in accordance with properties of events the server proposes the sequence flows with their possible splitting or fusion, see Figure 2. The sequence flows correspond to veins of plasmodia between active points (attractants).

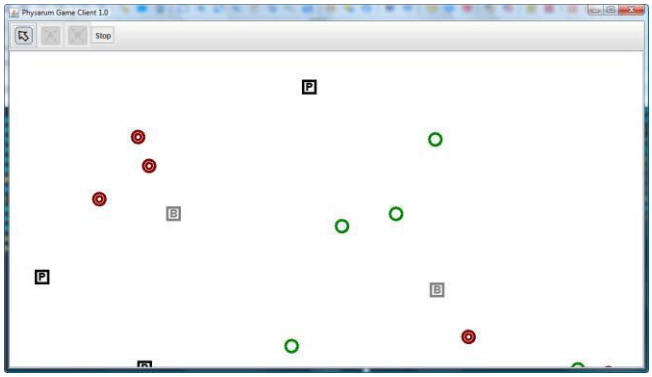

Figure 1: The *Physarum* client window for fixing events

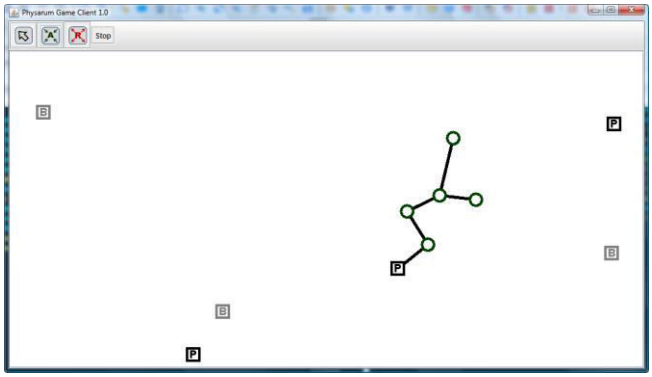

Figure 2: The *Physarum* client window with three sequence flows and one splitting

In this graphical notations, we use events and arcs among events. Formally, this means that we have a transition system $TS = (E, L, T, I_{in}, I_{end})$, where:

- E is the non-empty set of events,
- L is the set of labels for transitions,
- $T \subseteq E \times L \times E$ is the transition relation,
- $I_{in} \subseteq E$ is the set of initial events.
- $I_{end} \subseteq E$ is the set of final events.

This transition system $TS = (E, L, T, I_{in}, I_{end})$ is presented in our software tool as a labelled graph with nodes corresponding to events from E, edges representing the transition relation T, and labels of edges corresponding to members from L.

In the business process simulation, an event from E can be (i) a start event from I_{in} (to start a process), (ii) an end event from I_{end} (to finish a process), (iii) an intermediate message or timer event from $E \setminus (I_{in} \cup I_{end})$. A label from L is a business task interpreted as an atomic activity that has to be performed within a process. In the *Physarum* business process diagram language we can define several task types like that: service, receive, send, user, script, manual, and reference. A transition from T is understood as a gateway, i.e., a routing construct used to control the divergence and convergence of sequence flow.

A topological structure of the plasmodium motions can be described as a triple $\mathrm{PM} = (P, A, R)$, where

- $P = \{p_1, p_2, \ldots, p_k\}$ is a set of original points of plasmodia of *Physarum polycephalum*,
- $A = \{a_1, a_2, \ldots, a_m\}$ is a set of attractants,
- $R = \{r_1, r_2, \ldots, r_n\}$ is a set of repellents.

A behavior of plasmodia is described by the set $V = \{v_1, v_2, \ldots, v_r\}$ of protoplasmic veins formed among attractants as well as from original points of plasmodia to attractants.

The business processes of $TS = (E, L, T, I_{in}, I_{end})$ can be reformulated in the form of plasmodium transitions due to the following set of bijective functions:

- $\sigma : P \cup A \to E$ assigning an event to each original point of plasmodium as well as to each attractant,
- $\tau : V \to T$ assigning a transition to each protoplasmic vein,
- $\iota_{in} : P \to I_{in}$ assigning an initial event to each original point of plasmodium.
- $\iota_{end} : S \to I_{end}$ assigning an end event to each final point of plasmodium from $S \subseteq E$.

In $TS = (E, L, T, I_{in}, I_{end})$, there are different compositions of transitions from T that are called *gateways*:

- *parallel fork gateways* for creating concurrent sequence flows which have different end events;
- *parallel join gateways* AND for synchronizing all concurrent sequence flows: at the beginning they have a splitting of one sequence flow A into several flows $A_1, \ldots, A_n$, then a fusion AND($A_1, \ldots, A_n$) = B;
- *parallel join gateways* OR for synchronizing some concurrent sequence flows: at the beginning they have a splitting of one sequence flow A into several flows $A_1, \ldots, A_n$, then a fusion OR($A_1, \ldots, A_n$) = B;
- *event-based decision gateways* XOR: at the beginning we choose one alternative of a set of mutually exclusive sequence flows $A_1, \ldots, A_n$, then we join alternative sequence flows into one sequence flow XOR($A_1, \ldots, A_n$) = B.

Hence, a workflow net is based on the following items:

- the initial events of I_{in};
- the final events of I_{end};
- the set of tasks $Ts = E \setminus (I_{in} \cup I_{end})$;
- the flow relation $F \subseteq (I_{in} \times Ts) \cup (Ts \times I_{end}) \cup (Ts \times Ts)$ such that every node is on a directed path from $i \in I_{in}$ to $i \in I_{end}$;
- the splitting AND, OR, or XOR;
- the joining AND, OR, or XOR;
- the removing defines the subnet of the whole workflow that is cleansed when the task is executed;
- the definition of instances number of each task in accordance with functions MIN (the minimum number of instances of task $t \in Ts$), MAX (the maximum number of instances of $t \in Ts$), threshold (if the threshold is reached, then all active instances are cancelled and the task is considered completed).

Thus, in this classical definition of workflow nets we assume that we always can build up a workflow net, where every event and every transition is located on a path from the initial event to the

final event. So, the initial event $i \in I_{in}$ has no incoming edges and the final event $i \in I_{end}$ has no outgoing edges. After the splitting AND, OR, or XOR we can construct an appropriate fusion AND, OR, or XOR. Therefore we deal with a directed graph from $i \in I_{in}$ to $i \in I_{end}$.

However, in case of swarms like the plasmodium of *Physarum polycephalum* we face the situation that there are no directed graphs, because any swarm has a strategy to be expended in all accessible directions and, thus, to prefer the splitting instead of the fusion. In other words, we ever face more outputs $i \in I_{end}$ than inputs $i \in I_{in}$.

Another problem is that any swarm has free will and under the same conditions swarms can behave differently.

In our *Physarum* business process diagram language, we have implemented some features of swarms including their expansion strategies. As a result, we avoid atomic actions because of impossibility to construct a workflow net inductively on the basis of atomic tasks.

3. Reflexive Management Based on Swarm Behaviour

The *reflexive management* is used for controlling a knowledge structure of agents in a way such that all performations satisfy the centre's goals, i.e., they are maximally favourable for this centre.

The substantional kinds of reflexive management are as follows:

- institutional management (modification of admissible sets of actions of all groups of agents);
- motivational management (modification of goal functions of concrete agents);
- informational management (modification of information which agents use in decision making).

Informational management is devided into the following kinds:

- informational regulating (purposeful influence on information about states of affairs);
- expert management (purposeful influence on information about models of decision making);
- active prognosis (purposeful spread of information about future values of parameters depending on states of affairs and actions of actors).

In our *Physarum* business process diagram language, we can formulate reflexive games and reflexive management due to some swarm properties we have implemented in the business process model.

Let us start with defining some classes of transitions that will be used for defining reflexive games. For each event $e \in E$ in the transition system TS, we can determine its direct successors and predecessors in the form:

- $Post(e) = \bigcup_{l \in L} Post(e,l)$, where $Post(e,l) = \{e' \in E : (e,l,e') \in T\}$, is the *set of all direct successors of the event* $e \in E$;
- $Pre(e) = \bigcup_{l \in L} Pre(e,l)$, where $Pre(e,l) = \{e' \in E : (e',l,e) \in T\}$, the *set of all direct predecessors of the state* $e \in E$.

If there exists the event $e \in E$ in the transition system TS such that $card(Post(s)) > 1$, where $card$ is the cardinality of the set, then TS is called a non-deterministic transition system. In non-deterministic transition systems, we deal with ambiguity of direct successors of some states, i.e., we have a splitting of events/tasks without their next fusion in accordance with AND, OR, XOR.

In non-deterministic transition systems, we can appeal to to rough approximation of sets defined in rough set theory (cf. [17]). Let $TS = (E, L, T, I_{in}, I_{end})$ be a transition system and $X \subseteq E$. The

lower predecessor anticipation $Pre_*(X)$ of X is given by

$$Pre_*(X) = \{e \in E : Post(e) \neq \emptyset \text{ and } Post(e) \subseteq X\}. \quad (1)$$

The lower predecessor anticipation consists of all events/tasks from which TS surely goes to the events/tasks in X.

The *upper predecessor anticipation* $Pre^*(X)$ of X is given by

$$Pre^*(X) = \{e \in E : Post(e) \cap X \neq \emptyset\}. \quad (2)$$

The upper predecessor anticipation consists of all events/tasks from which TS possibly goes to the events/tasks in X. So, TS can also go to the events/tasks from outside X.

In terms of the rough sets approach, called the Variable Precision Rough Set Model, we can reformulate an inclusion relation. Let U be a given set of elements and $A, B \subseteq U$. The standard set inclusion is defined as

$$A \subseteq B \text{ if and only if } \underset{u \in A}{\forall} u \in B. \quad (3)$$

Now, let U be a given set of elements, $A, B \subseteq U$, and $0 \leq \beta < 0.5$. The *majority set inclusion* is defined as

$$A \overset{\beta}{\subseteq} B \text{ if and only if } 1 - \frac{card(A \cap B)}{card(A)} \leq \beta, \quad (4)$$

where *card* denotes the cardinality of the set. $A \overset{\beta}{\subseteq} B$ means that a specified majority of elements belonging to A belongs also to B. One can see that, if $\beta = 0$, then the majority set inclusion becomes a standard set inclusion.

By replacing the standard set inclusion with the majority set inclusion in the original definition of the lower predecessor anticipation of a set of states in a transition system, we obtain the following generalized notion of the β-lower predecessor anticipation. Let $TS = (E, L, T, I_{in}, I_{end})$ be a transition system and $X \subseteq E$. The *β-lower predecessor anticipation* $Pre_*^{\beta}(X)$ of X is given by

$$Pre_*^{\beta}(X) = \{e \in E : Post(e) \neq \emptyset \text{ and } Post(e) \overset{\beta}{\subseteq} X\}. \quad (5)$$

The β-lower predecessor anticipation consists of each events/tasks from which TS goes, in most cases (i.e., in terms of the majority set inclusion) to the events/tasks in X.

Let $TS = (E, L, T, I_{in}, I_{end})$ be a transition system, $X \subseteq E$, and $0 \leq \beta < 0.5$. If $e \in Pre_*(X)$, then e is said to be a strict anticipator of events/tasks from X. If $e \in Pre_*^{\beta}(X)$, then e is said to be a quasi-anticipator of events/tasks from X. A set of all strict anticipators of X will be denoted by $\overline{Ant}(X)$ whereas a set of all quasi-anticipators of X will be denoted by $\overset{\approx}{Ant}(X)$. Notice that for any

X, $\overline{Ant}(X) \subseteq \overset{\approx}{Ant}(X)$.

Now we can extend a transition system $TS = (E, L, T, I_{in}, I_{end})$ of the *Physarum* business process diagram language to a *reflexive management language* $\mathsf{G} = (E, \mathsf{P}(E), Agt, Act, Mov, Tab, (\circ_A)_{A \in Agt_{1,2}})$, where

- E is a set of events/tasks of TS;
- $\mathsf{P}(E)$ is a set of payoffs in reflexive games;
- Agt is a set of reflexive players (agents);
- Act is a non-empty set of strategies represented by $\overline{Ant}(X)$ or $\overset{\approx}{Ant}(X)$ for each payoff $X \subseteq E$, an element of Act^{Agt} is called a move;
- $Mov : \mathsf{P}(E) \times Act^{Agt} \to 2^{Act} \setminus \{\varnothing\}$ is a mapping indicating the available sets of actions to a given player in a given set of events/tasks;
- $Tab : \mathsf{P}(E) \times Act^{Agt} \to \mathsf{P}(E)$ is the transition table which associates, with a given set of events/tasks of the game and a given move of the players, the set of events/tasks of the game resulting from that move;
- for each $A \in Agt$, $\circ_A$ is a preorder (reflexive and transitive relation) over subsets of E, called the *preference relation* of player A.

Thus, player's strategies $\overline{Ant}(X)$ or $\overset{\approx}{Ant}(X)$ are not exclusive, i.e., they can be intersected. The last one is the main feature of reflexive games and swarm behaviour. In these games we cannot define exclusive strategies at all. In reflexive games we can be engaged in an unlimited hierarchy of cognitive pictures: (i) each of the players can have their own picture about a state of affairs A, let us denote these pictures by $K_1 A$ and $K_2 A$; (ii) the first-order reflexion is expressed by means of pictures of the second order which are designated by $K_2 K_1 A$ and $K_1 K_2 A$, where $K_2 K_1 A$ are pictures of agent 2 about pictures of agent 1, and $K_1 K_2 A$ are pictures of agent 1 about pictures of agent 2, etc. In reflexive management we coose the level of reflexion n ($n > 0$) to coordinate actions of all players.

4. Conclusions

We have shown that communicating sequential processes formalized in standard business process diagram languages have ho emergent properties of swarms and if we try to implement these properties into business process diagram languages we face situations when splitting appears more often than fusion (section 1). After this implementation we can construct a reflexive management language (section 2) that contains some properties of swarms.

Acknowledgement

The research was financed by the National Science Centre in Poland, based on the decision DEC-2012/07/B/HS1/00263, and by the Seventh Framework Programme, FP7-ICT-2011-8.

References:

1. Aalst, W.M.P. van der, Hofstede, A.H.M. ter, Kiepuszewski, B., and Barros, A. P. Workflow Patterns, *Distributed and Parallel Databases*, 14(3), July 2003, 5-51.
2. Adamatzky, A. *Physarum Machines: Computers from Slime Mould*. World Scientific, 2010.
3. Adamatzky, A., Erokhin, V., Grube, M., Schubert, Th., Schumann A. Physarum Chip Project: Growing Computers From Slime Mould, *International Journal of Unconventional Computing*, 8(4), 2012, 319-323.
4. Ammarguellat, Z. A control-flow normalization algorithm and its complexity. *IEEE Trans. Software Eng*., 18(3), 1992, 237-251.
5. Becker, J., Kugeler, M., and Rosemann, M. (eds.). Process Management. *A Guide for the Design of Business Processes*. Springer-Verlag, 2003.
6. Brambilla, M., Ceri, S., Fraternali, P., and Manolescu, I. Process modeling in web applications, *ACM Trans. Softw. Eng. Methodol.*, 15(4), 2006, 360-409.
7. Desel, J., and Esparza, J. (eds.). *Free Choice Petri Nets, volume 40 of Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
8. Gardner, T. UML Modelling of Automated Business Processes with a Mapping to BPEL4WS. In: *Proceedings of the First European Workshop on Object Orientation and Web Services*. Springer, 2003.
9. Johnson, R., Pearson, D., and Pingali, K. The Program Structure Tree: Computing control regions in linear time. [in:] *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Orlando FL, USA, ACM Press, June 1994, 171-185.
10. Jordan, D., and Evdemon, J. *Web Services Business Process Execution Language Version 2.0. Committee Specification*. OASIS WS-BPEL TC, January 2007. Available via http://www.oasis-open.org/committees/download.php/22475/wsbpel-v2.0-CS01.pdf.
11. Kiepuszewski, B., Hofstede, A.H.M. ter, and Aalst, W.M.P. van der. Fundamentals of control flow in workflows. *Acta Informatica*, 39(3), 2003, 143-209.
12. Pancerz, K., Schumann, A. Principles of an Object-Oriented Programming Language for Physarum Polycephalum Computing. [in:] *Proceedings of the 10th International Conference on Digital Technologies* (DT'2014), Zilina, Slovak Republic, July 9-11, 2014, 273-280.
13. Rosemann, M. Preparation of Process Modeling. In J. Becker, M. Kugeler, and M. Rosemann, (eds.). *Process Management. A Guide for the Design of Business Processes*, pages 41–78. Springer-Verlag, 2003.
14. Schumann, A. Towards context-based concurrent formal theories, *Parallel Processing Letters*, 25, 2015, 1540008.
15. Schumann, A., Pancerz, K. Towards an Object-Oriented Programming Language for Physarum Polycephalum Computing. [in:] Szczuka, M., Czaja, L., Kacprzak, M. (eds.). *Proceedings of the Workshop on Concurrency, Specification and Programming* (CS&P'2013), Warsaw, Poland, September 25-27, 2013, 389-397.
16. Schumann, A., Pancerz, K. Timed Transition System Models for Programming Physarum Machines: Extended Abstract. [in:] Popova-Zeugmann, L. (ed.). *Proceedings of the Workshop on Concurrency, Specification and Programming* (CS&P'2014), Chemnitz, Germany, September 29 - October 1, 2014.
17. Schumann, A., Pancerz, K. Roughness in Timed Transition Systems Modeling Propagation of Plasmodium. [in:] Ciucci, D. et al. (eds.). *Rough Sets and Knowledge Technology*, *Lecture Notes in Computer Science*, 9436, 2015.
18. Zhao, W., Hauser, R., Bhattacharya, K., Bryant, B.R., and Cao, F. Compiling business processes:

untangling unstructured loops in irreducible flow graphs, *Int. Journal of Web and Grid Services*, 2(1), 2006, 68-91.