Functional Stream Derivatives of Context-Awareness on P2P Networks

Phan Cong Vinh[1], Nguyen Thanh Tung[2], Nguyen Van Phuc[1], and Nguyen Hai Thanh[2]

[1] Department of IT, NTT University, 300A Nguyen Tat Thanh St., Ward 13, District 4, HCM City, Vietnam
pcvinh@ntt.edu.vn, phuc.tsu@gmail.com

[2] International School, Vietnam National University in Hanoi, 144 Xuan Thuy St., Cau Giay District, Hanoi, Vietnam
{tungnt,thanh.ishn}@isvnu.vn

Abstract. This paper will be both to give an in-depth analysis as well as to present the new material on the notion of context-awareness process on P2P networks, an idea that networking can both sense and react accordantly based on external actions. The paper formalizes context-awareness process using the notion of *functional stream derivative*, including P2P networks, context-awareness and the functional stream derivatives of context-awareness on P2P networks. A brief summary of this approach is also given.

Keywords: Context-awareness, Context-awareness process, Functional stream derivative, P2P networks.

1 Introduction

In development of P2P networks, one of the limitations of the current approaches is that when increasing (fully or partially) the context-awareness of networking, the semantics and understanding of the context-awareness process become difficult to capture for the development. As motivation, the context-awareness process on P2P networks should be carefully considered under a suitably rigorous mathematical structure to capture its semantics completely, and then support an automatic developing process, in particular, and applications of context-aware networking, generally.

Both initial algebras and final coalgebras are mathematical tools that can supply abstract representations to aspects of the context-awareness process on P2P networks. On the one hand, algebras can specify the operators and values. On the other hand, coalgebras, based on a collection of observers, are considered in this paper as a useful framework to model and reason about the context-awareness process on P2P networks. Both initiality and finality give rise to a basis for the development of context-awareness calculi on P2P networks directly based on and driven by the specifications. From a programming point of view, this paper provides coalgebraic structures to develop the applications in the area of context-aware computing on P2P networks.

A coalgebraic structure provides an expressive, powerful and uniform view of context-awareness, in which the observation of context-awareness processes on P2P networks plays a central role. The concepts of bisimulation and homomorphism of context-awareness are used to compute the context-awareness process on P2P networks.

2 Outline

The paper is a reference material for readers who already have a basic understanding of P2P networks and are now ready to know the novel approach for formalizing context-awareness process on such P2P networks using coalgebraic language.

Formalization is presented in a straightforward fashion by discussing in detail the necessary components and briefly touching on the more advanced components. The notion of functional stream derivatives, including justifications needed in order to achieve the particular results, is also presented.

The rest of this paper is organized as follows: Section 3 briefly describes related work and existing concepts. P2P networks and context-awareness are the subjects of Section 4. Section 5 presents functional stream derivatives of context-awareness. Finally, Section 6 is a brief summary.

3 Related Work and Existing Concepts

In our previous paper [15], we have rigorously approached to the notion of context-awareness in context-aware systems from which coalgebraic aspects of the context-awareness emerge. The coalgebraic model is used to formalize the unifying frameworks for context-awareness and evolution of the context-awareness processes, respectively.

Most notions and observations of this paper are instances of a theory called universal coalgebra [10,4]. In [9,11], some recent developments in coalgebra are presented.

The programming paradigm with functions called functional programming [1,5,2,3,7] treats computation as the evaluation of mathematical functions. Functional programming emphasizes the evaluation of functional expressions. The expressions are formed by using functions to combine basic values.

The notion of bisimulation is a categorical generalization that applies to many different instances of infinite data structures, various other types of automata, and dynamic systems [10,9,4]. In theoretical computer science, a bisimulation is an equivalence relation between abstract machines, also called the abstract computers or state transition systems (i.e., a theoretical model of a computer hardware or software system) used in the study of computation. Abstraction of computing is usually considered as discrete time processes. Two computing systems are bisimular if, regarding their behaviors, each of the systems “simulates” the other and vice-versa. In other words, each of the systems cannot be distinguished from the other by the observation.

Homomorphism is one of the fundamental concepts in abstract algebra [8], which scrutinizes the sets of algebraic objects, operations on those algebraic objects, and functions from one set of algebraic objects to another. A function that preserves the operations on the algebraic objects is known as a homomorphism. In other words, if an algebraic object includes several operations, then all its operations must be preserved for a function to be a homomorphism in that category [13,6].

4 P2P Networks and Context-Awareness

4.1 P2P Networks

A network, which consists of the set of peers (considered as nodes) together with morphisms $\_ \parallel \_$ in the set of parallel compositions (considered as edges), generates P2P structure [14]. The P2P structure is dynamic in nature because peers can be dynamically added to or dropped from the network. For such every action, *context-awareness* for the P2P structure occurs.

4.2 Context-Awareness

Let PEER be the set of peers and $\mathsf{SYS}= \{\parallel_{i\in\mathbb{N}_0} a_i \text{ with } a_i \in \mathsf{PEER}\}$ be the set of parallel compositions on the P2P network.
Let $T = \{add, drop\}$ be the set of actions making a P2P structure on the network change, in which *add* and *drop* are defined as follows:

add is a binary operation

$$add : \mathsf{SYS} \times \mathsf{PEER} \longrightarrow \mathsf{SYS} \tag{1}$$

(sometimes specified as $\mathsf{SYS} \xrightarrow{add(\mathsf{PEER})} \mathsf{SYS}$ or $add(\mathsf{PEER}) : \mathsf{SYS} \longrightarrow \mathsf{SYS}$)

obeying the following axioms: For all $i \in \mathbb{N}_0$,

$$add(\parallel_i a_i, b) = \begin{cases} (\parallel_{1\leqslant i\leqslant n} a_i) \parallel b & \text{for } i \geqslant 1 \\ (\parallel_0) \parallel b = skip \parallel b = b & \text{when } i = 0 \end{cases} \tag{2}$$

or, also written as

$$\begin{cases} \parallel_{1\leqslant i\leqslant n} a_i \xrightarrow{add(b)} (\parallel_{1\leqslant i\leqslant n} a_i) \parallel b & \text{for } i \geqslant 1 \\ \parallel_0 \xrightarrow{add(b)} (\parallel_0) \parallel b = skip \parallel b = b & \text{when } i = 0 \end{cases}$$

or

$$\begin{cases} add(b) : \parallel_{1\leqslant i\leqslant n} a_i \longrightarrow (\parallel_{1\leqslant i\leqslant n} a_i) \parallel b & \text{for } i \geqslant 1 \\ add(b) : \parallel_0 \longrightarrow (\parallel_0) \parallel b = skip \parallel b = b & \text{when } i = 0 \end{cases}$$

Example:

$$\begin{aligned} &add(\|_0, a) &&= a \\ &add(a, b) &&= a \parallel b \\ &add(a \parallel b, c) &&= a \parallel b \parallel c \end{aligned}$$

drop is also a binary operation

$$drop : \mathsf{SYS} \times \mathsf{PEER} \longrightarrow \mathsf{SYS} \tag{3}$$

(sometimes specified as $\mathsf{SYS} \xrightarrow{drop(\mathsf{PEER})} \mathsf{SYS}$ or $drop(\mathsf{PEER}) : \mathsf{SYS} \longrightarrow \mathsf{SYS}$)

obeying the following axioms: For all $i \in \mathbb{N}_0$,

$$drop(\|_i\ a_i, b) = \begin{cases} \|_{1 \leqslant i \leqslant (n-1)}\ a_i & \text{when there exists } a_i = b \\ \|_{1 \leqslant i \leqslant n}\ a_i & \text{for all } a_i \neq b \end{cases} \tag{4}$$

or, also written as

$$\begin{cases} \|_{1 \leqslant i \leqslant n}\ a_i \xrightarrow{drop(b)} \|_{1 \leqslant i \leqslant (n-1)}\ a_i & \text{when there exists } a_i = b \\ \|_{1 \leqslant i \leqslant n}\ a_i \xrightarrow{drop(b)} \|_{1 \leqslant i \leqslant n}\ a_i & \text{for all } a_i \neq b \end{cases}$$

or

$$\begin{cases} drop(b) & : \|_{1 \leqslant i \leqslant n}\ a_i \longrightarrow \|_{1 \leqslant i \leqslant (n-1)}\ a_i & \text{when there exists } a_i = b \\ drop(b) & : \|_{1 \leqslant i \leqslant n}\ a_i \longrightarrow \|_{1 \leqslant i \leqslant n}\ a_i & \text{for all } a_i \neq b \end{cases}$$

It follows that $drop(\|_0, b) = \|_0 = skip$.

Example:

$$\begin{aligned} &drop(a, a) &&= \|_0 \\ &drop(a \parallel b \parallel c, b) &&= a \parallel c \\ &drop(a \parallel b \parallel c, d) &&= a \parallel b \parallel c \end{aligned}$$

A context-awareness process is completely defined when actions *add* and *drop* are executed on a P2P network as illustrated in the following diagram:

$$\begin{tikzcd}
(\|_0) \arrow[r, bend left, "add"] \arrow[loop right, "drop"] & (a_1) \arrow[l, bend left, "drop"] \arrow[r, bend left, "add"] \arrow[loop right, "drop"] & (a_1 \parallel a_2) \arrow[l, bend left, "drop"] \arrow[r, bend left, "add"] \arrow[loop right, "drop"] & (a_1 \parallel a_2 \parallel a_3) \arrow[l, bend left, "drop"] \arrow[r, bend left, "add"] \arrow[loop right, "drop"] & \cdots \arrow[l, bend left, "drop"]
\end{tikzcd} \tag{5}$$

In consideration of P2P networks, context-awarenesses are known as *homomorphisms* from a P2P network to another P2P network to preserve the P2P structure. In other words, context-awareness is a map from a set of parallel compositions to another set of parallel compositions of the same type that preserves all the P2P structures.

Definition 1 (Context-Awareness). *Let $T = \{add, drop\}$ be a set of actions. A context-awareness with set of actions T is a pair $\langle \mathsf{SYS}, \langle o_{\mathsf{SYS}}, e_{\mathsf{SYS}} \rangle \rangle$ consisting of*

- *a set SYS of* P2P networks*,*
- *an* output function $o_{\mathsf{SYS}} : \mathsf{SYS} \longrightarrow (T \longrightarrow \mathit{2})$*, and*
- *an* evolution function $e_{\mathsf{SYS}} : \mathsf{SYS} \longrightarrow (T \longrightarrow \mathsf{SYS})$.

where

- $\mathbf{2} = \{0, 1\}$,
- o_{SYS} assigns, to a network c, a function $o_{\mathsf{SYS}}(c) : T \longrightarrow \mathbf{2}$, which specifies the value $o_{\mathsf{SYS}}(c)(t)$ that is reached after an action t has been executed. In other words,

$$o_{\mathsf{SYS}}(c)(t) = \begin{cases} 1 \text{ when } t \text{ becomes fully available, or} \\ 0 \text{ otherwise} \end{cases}$$

- Similarly, e_{SYS} assigns, to a network c, a function $e_{\mathsf{SYS}}(c) : T \longrightarrow \mathsf{SYS}$, which specifies the network $e_{\mathsf{SYS}}(c)(t)$ that is reached after an action t has been executed. Sometimes $c \xrightarrow{t} c'$ is used to denote $e_{\mathsf{SYS}}(c)(t) = c'$.

Generally, both the network space SYS and the set T of actions may be infinite. If both SYS and T are finite, then we have a finite context-awareness, otherwise we have an infinite context-awareness.

5 Functional Stream Derivatives of Context-Awareness

The notion of *functional stream derivative* is Rutten's new contribution in [12]. This concept is defined for functions on streams over arbitrary inputs. Hence if the operators of any algebra of stream functions can be defined by notion of *stream differential equations* then all of them are available to apply Rutten's functional stream derivative.

Here the application of this functional stream derivative is found in our algebra of context-awareness involving two actions: *add* and *drop*. Therefore, a network can be changed to become another one in the set of parallel compositions $\mathsf{SYS} = \{\|_{i \in \mathbb{N}_0} a_i\}$ (see diagram 5). In other words, the set of parallel compositions SYS is closed under the actions in T. Below are some basic Rutten's stream concepts [12] manipulated on context-awareness.

- A *stream* over a set of actions T is an infinite sequence of consecutive actions in T, obtained by repeatedly applying the evolution function e_{SYS}. Let σ denote a stream that is described as

$$\sigma = (\sigma(0), \sigma(1), \sigma(2), ...) \ : \ \{0, 1, 2, ...\} \longrightarrow T$$

- A set of *streams*, denoted by T^ω, over a set of actions T is determined by $T^\omega = \{\sigma \mid \sigma : \{0,1,2,...\} \longrightarrow T\} = \{(\sigma(0),\sigma(1),\sigma(2),...) \mid (\sigma(0),\sigma(1),\sigma(2),...) : \{0,1,2,...\} \longrightarrow T\}$. Similarly, another set of streams $\mathbf{2}^\omega$ over the boolean set $\mathbf{2} = \{0,1\}$ is determined by $\mathbf{2}^\omega = \{\sigma \mid \sigma : \{0,1,2,...\} \longrightarrow \mathbf{2}\} = \{(\sigma(0),\sigma(1),\sigma(2),...) \mid (\sigma(0),\sigma(1),\sigma(2),...) : \{0,1,2,...\} \longrightarrow \mathbf{2}\}$
- The *stream derivative* of a stream σ is defined by $\sigma' = (\sigma(1),\sigma(2),\sigma(3),...)$ and $\sigma(0)$ is called the *initial value* of σ.
- A function $f : T^\omega \longrightarrow \mathbf{2}^\omega$ is called *causal* if for any $\sigma,\tau \in T^\omega$, $i \geqslant 0, t_0,...,t_i \in T$, and with $t_0 : ... : t_i : \sigma$ denoting $(t_0,...,t_i,\sigma(0),\sigma(1),...)$, then

$$f(t_0 : ... : t_i : \sigma)(i) = f(t_0 : ... : t_i : \tau)(i)$$

  That is, the i^{th} element of the stream $f(\sigma)$ depend only on the i elements early in σ.
- For a causal function $f : T^\omega \longrightarrow \mathbf{2}^\omega$, action $t \in T$ and $\sigma \in T^\omega$,
  - (i) The initial output of f is defined by $f(t : \sigma)(0)$ and denoted by $f[t]$.
  - (ii) The functional stream derivative of f, denoted as $f_t : T^\omega \longrightarrow \mathbf{2}^\omega$, is defined by a causal function $f_t(\sigma) = f(t : \sigma)'$. That is, f_t acts as f on the rest of σ after the first t.

Let f be causal and $\Gamma = \{f \mid f : T^\omega \longrightarrow \mathbf{2}^\omega\}$. For f in Γ and t in T, the function $\langle o_\Gamma, e_\Gamma\rangle : \Gamma \longrightarrow (\mathbf{2} \times \Gamma)^T$ is defined by $\langle o_\Gamma, e_\Gamma\rangle = \langle f[t], f_t\rangle$. This notion offers a context-awareness process $\langle \Gamma, \langle o_\Gamma, e_\Gamma\rangle\rangle$ with evolution as follows:

$$f \xrightarrow{t|f[t]} f_t$$

As mentioned in our other publication [15], a homomorphism between $\langle \mathsf{SYS}, \langle o_{\mathsf{SYS}}, e_{\mathsf{SYS}}\rangle\rangle$ and $\langle \Gamma, \langle o_\Gamma, e_\Gamma\rangle\rangle$ is any function $h : \mathsf{SYS} \longrightarrow \Gamma$ such that

$$c \xrightarrow{t|o_{\mathsf{SYS}}(c)(t)} e_{\mathsf{SYS}}(c)(t) \Longrightarrow h(c) \xrightarrow{t|o_\Gamma(h(c))(t)} e_\Gamma(h(c))(t)$$

Theorem 1. *The context-awareness $\langle \Gamma, \langle o_\Gamma, e_\Gamma\rangle\rangle$ is* final *among all context-awarenesses; i.e., for any context-awareness $\langle \mathsf{SYS}, \langle o_{\mathsf{SYS}}, e_{\mathsf{SYS}}\rangle\rangle$, there* exists *a* unique *homomorphism $h : \langle \mathsf{SYS}, \langle o_{\mathsf{SYS}}, e_{\mathsf{SYS}}\rangle\rangle \longrightarrow \langle \Gamma, \langle o_\Gamma, e_\Gamma\rangle\rangle$.*

Proof. For $\langle \mathsf{SYS}, \langle o_{\mathsf{SYS}}, e_{\mathsf{SYS}}\rangle\rangle$, we define a function $h : \langle \mathsf{SYS}, \langle o_{\mathsf{SYS}}, e_{\mathsf{SYS}}\rangle\rangle \longrightarrow \langle \Gamma, \langle o_\Gamma, e_\Gamma\rangle\rangle$, whose commutative diagram is as follows:

$$\begin{tikzcd}
\mathsf{SYS} \arrow[r, "f=\langle o_{\mathsf{SYS}}, t_{\mathsf{SYS}}\rangle"] \arrow[d, "h"'] & \odot\ \mathsf{SYS} \arrow[d, "\odot\ h"] \\
\Gamma \arrow[r, "\omega=\langle o_\Gamma, t_\Gamma\rangle"] & \odot\ \Gamma
\end{tikzcd}$$

In other words, for c in SYS, we must define a function $h(c) : T^\omega \longrightarrow \mathbf{2}^\omega$ by considering, for σ in T^ω and $n \geqslant 0$, a stream of consecutive actions in T obtained by repeatedly applying the evolution function e_{SYS}.

$$c \xrightarrow{\sigma(0)|o_{\mathsf{SYS}}(c)(\sigma(0))} c_1 \xrightarrow{\sigma(1)|o_{\mathsf{SYS}}(\sigma(1))(t)} \ldots \xrightarrow{\sigma(n)|o_{\mathsf{SYS}}(c_n)(\sigma(n))} c_{n+1} \cdots$$

and assigning $h(c)(\sigma(n)) = o_{\mathsf{SYS}}(c_n)(\sigma(n))$. In this way, h is a homomorphism and unique. In addition, $h(c)$ is also causal.

The stream function $h(c)$ is called the context-awareness process of c. For a causal function f in Γ, network c in SYS is called an *implementation* of f if $f = h(c)$ [12].

From the universal view, for a network c in SYS of $\langle \mathsf{SYS}, \langle o_{\mathsf{SYS}}, e_{\mathsf{SYS}} \rangle \rangle$, $\langle c \rangle \subseteq \langle \mathsf{SYS}, \langle o_{\mathsf{SYS}}, e_{\mathsf{SYS}} \rangle \rangle$ denotes the smallest subset containing c and closed under evolutions for any actions in T. Hence $\langle c \rangle$ is also a subcontext-awareness of $\langle \mathsf{SYS}, \langle o_{\mathsf{SYS}}, e_{\mathsf{SYS}} \rangle \rangle$ generated from c by applying its evolution function e_{SYS} restrictively over the set $\langle c \rangle$.

As considered in [12], the two following corollaries are also true for our context-awareness.

Corollary 1. *Subcontext-awareness $\langle f \rangle \subseteq \langle \Gamma, \langle o_\Gamma, e_\Gamma \rangle \rangle$ implements any causal function f in Γ.*

Proof. As in [12], this is trivial because it follows that the inclusion function $i : \langle f \rangle \longrightarrow \Gamma$ (i.e., $i(f) = f$) is homomorphism.

Corollary 2. *$\langle f \rangle$ is a minimal context-awareness process containing the smallest number of networks that implements f.*

Proof. As in [12], for context-awareness $\langle \langle c \rangle, \langle o_{\langle c \rangle}, e_{\langle c \rangle} \rangle \rangle$ and h in theorem 1, the network c in $\langle c \rangle$ implements f; therefore $h(c) = f$. Because h is a homomorphism, it implies that $h(\langle c \rangle) = \langle h(c) \rangle = \langle f \rangle$. Hence the size of $\langle h(c) \rangle$ is the size of $\langle f \rangle$.

6 Conclusions

In this paper, the notion of Rutten's functional stream derivatives is used to formalize the evolution of the context-awareness process on P2P networks. In addition, as our future work, based on functional stream derivatives, a coalgebraic implementation procedure of context-awareness on P2P networks can also be developed.

Acknowledgements. Thank you to NTTUFSTD[1] for the constant support of our work which culminated in the publication of this paper. As always, we are deeply indebted to the anonymous reviewers for their helpful comments and valuable suggestions which have contributed to the final preparation of the paper.

[1] The NTTU foundation for Science and Technology Development.

References

1. Barbosa, L.S.: Components as Processes: An Exercise in Coalgebraic Modeling. In: Smith, S.F., Talcott, C.L. (eds.) 4th International Conference on Formal Methods for Open Object-Based Distributed Systems, IFIP TC6/WG6.1, Stanford, CA, USA, September 6-8, pp. 397–417. Kluwer Academic Publishers (2000)
2. Cockett, R., Spencer, D.: Strong Categorical Datatypes I. In: Seely, R.A.G. (ed.) International Summer Meeting on Category Theory, Montréal, Québec, Canada, June 23-30, pp. 141–169. AMS Canadian Mathematical Society (1991)
3. Hagino, T.: A Typed Lambda Calculus with Categorical Type Constructors. In: Pitt, D.H., Rydeheard, D.E., Poigné, A. (eds.) Category Theory and Computer Science. LNCS, vol. 283, pp. 140–157. Springer, Heidelberg (1987)
4. Jacobs, B., Rutten, J.: A Tutorial on (Co)Algebras and (Co)Induction. Bulletin of EATCS 62, 222–259 (1997)
5. Kieburtz, R.B.: Reactive Functional Programming. In: Gries, D., de Roever, W.P. (eds.) Programming Concepts and Methods (PROCOMET), IFIP International Federation for Information Processing, Shelter Island, NY, USA, June 8-12, pp. 263–284. Chapman and Hall (1998)
6. Levine, M.: Categorical Algebra. In: Benkart, G., Ratiu, T.S., Masur, H.A., Renardy, M. (eds.) Mixed Motives. Mathematical Surveys and Monographs, vol. 57, I, II, II, Part II, pp. 373–499. American Mathematical Society, USA (1998)
7. Meijer, E., Fokkinga, M., Paterson, R.: Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire. In: Hughes, J. (ed.) FPCA 1991. LNCS, vol. 523, pp. 124–144. Springer, Heidelberg (1991)
8. Rotman, J.J.: Advanced Modern Algebra, 1st edn. Prentice Hall, USA (2002)
9. Rutten, J.J.M.M.: Automata and Coinduction (an Exercise in Coalgebra). In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 194–218. Springer, Heidelberg (1998)
10. Rutten, J.J.M.M.: Universal Coalgebra: A Theory of Systems. Theoretical Computer Science 249(1), 3–80 (2000)
11. Rutten, J.J.M.M.: Algebra, Bitstreams, and Circuits. Technical Report SEN-R0502, CWI, Amsterdam, The Netherlands (2005)
12. Rutten, J.J.M.M.: Algebraic Specification and Coalgebraic Synthesis of Mealy Automata. In: Barbosa, L.S., Liu, Z. (eds.) 2nd International Workshop on Formal Aspects of Component Software (FACS), UNU/IIST, Macao, October 24-25, ENTCS (2005)
13. van Oosten, J.: Basic Category Theory. Department of Mathematics, Utrecht University, The Netherlands (July 2002)
14. Vinh, P.C.: Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development, and Verification. In: Formal Specification and Verification of Self-Configuring P2P Networking: A Case Study in Mobile Environments, 1st edn., pp. 170–188. IGI Global (2011)
15. Vinh, P.C., Tung, N.T.: Coalgebraic aspects of Context-Awareness. Mobile Networks and Applications (August 2012), doi:10.1007/s11036-012-0404-0