Computational Environment of Software Agents[1]

Martin Tomášek

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice
Letná 9, 042 00 Košice, Slovakia
E-mail: martin.tomasek@tuke.sk

Abstract: Presented process calculus for software agent communication and mobility can be used to express distributed computational environment and mobile code applications in general. Agents are abstraction of the functional part of the system architecture and they are modeled as process terms. Agent actions model interactions within the distributed environment: local/remote communication and mobility. Places are abstraction of the single computational environment where the agents are evaluated and where interactions take place. Distributed environment is modeled as a parallel composition of places where each place is evolving asynchronously. Operational semantics defines rules to describe behavior within the distributed environment and provides a guideline for implementations. Via a series of examples we show that mobile code applications can be naturally modeled.

Keywords: software agent, multi-agent system, mobile code, communication, process calculus, mobile agent

1 Introduction

Mobile agent [1] is an autonomous program that decides which places of the distributed application visits and what operations uses there. Distributed systems based on mobile agents are more flexible than static ones: they support mobile users and can reduce network bandwidth [2]. It means the user just sends an agent then disconnects from network and finally receives the agent with result upon new connection.

Formal description and specification of such systems is very important for modeling and successful implementation of the application. If we think of most important system characteristics, we identify communication and mobility as a key point. There are a lot of techniques to describe mobile processes and

[1] This paper was supported by the grant Nr. 1/0176/03 of the Slovak Grant Agency.

communication in existence. Very powerful tools for describing parallelism, communication and mobility are process algebras [3] and other formal techniques [4].

In this paper we present process calculus to describe mobile agents and their communication strategies. We provide basic abstraction of the distributed system and its parts and we define syntactic and semantics rules for modeling mobile applications. At the end we provide a formal description of three mobile code paradigms to illustrate the flexibility and expressiveness of the presented abstraction. Some very typical applications that implement code mobility are showed too.

2 Abstraction of the Software Architecture

We can identify three main entities from the abstraction of distributed system architecture: agents, interactions and places.

Agents are abstraction of the functional part of the system. They are evaluated in distributed computational environment and they are performing basic actions in their evolution.

Interactions are events presented between two agents or more agents in the computational environment. Basic agent actions are communication and mobility.

Places are abstraction of distributed computational environment. Whole distributed system is a set of places. Each place consists of agents and they are evaluated there. Interactions between agents can rise within one place or between two or more places.

3 Abstract Syntax of the System

We define terms of process algebra for modeling agents that can interact by performing three basic actions (read, write and move). The agents are modeled as process terms. The constructions for building agent terms are taken from Milner's CCS [5] and π-calculus [6, 7] and correspond to basic notions of process algebras [3].

Distributed system is defined as a parallel composition of independent places within a network. Each place is represented by its name and an agent term defining agents located inside the place. We define operator || for parallel composition of places and its notion is very similar to | operator for parallel composition of agents.

Abstract syntax of the calculus is following:

α ::=		(actions)
|	x	(perform name)
|	$\mathbf{r}_p(x)$	(read name)
|	$\mathbf{w}_p(y)$	(write name)
|	$\mathbf{m}_p(Q)$	(move agent)
P ::=		(agents)
|	$\mathbf{0}$	(inactivity)
|	$\alpha.P$	(action composition)
|	$P_1 \mid P_2$	(parallel composition)
|	$P_1 + P_2$	(choice)
|	$A\langle\tilde{x}\rangle$	(agent invocation)
S ::=		(system)
|	$[P]_p$	(place)
|	$S_1 \parallel S_2$	(system composition)

Symbols $x, y, p, \ldots$ are called names and $\mathcal{N}$ is the set of all names. Names are an abstraction of manipulated data within agent interactions. Abbreviation $\tilde{x}$ is a sequence of names and $\{\tilde{x}\}$ is a set of names in $\tilde{x}$.

Symbol α denominates the actions provided by the agents. Action x performs an operation represented by name x. Action $\mathbf{r}_p(x)$ reads a name that was sent by another agent to place p and stores it in name x. Action $\mathbf{w}_p(x)$ outputs name x in place named as p. Action $\mathbf{m}_p(P)$ moves agent term P to the place p and the term P is computed there.

Agents are defined as process terms very similar way as in other standard calculi and they are denominated as $P, Q, \ldots$ symbols. The inactivity $\mathbf{0}$ defines an agent with no activity. Term $\alpha.P$ is an action composition and its notion is that when an action α is performed the term continues as P. Parallel composition $P_1 \mid P_2$ defines two independent agents P_1 and P_2 that can be computed in parallel. Agent term $P_1 + P_2$ is nondeterministic choice where an agent can be computed either as P_1 or P_2.

We assume that each agent abstraction A is defined by equation $A(\tilde{x}) \stackrel{def}{=} P_A$ where all free names of P_A are contained in $\tilde{x}$. Process abstraction is then a term

without free names while $A(\tilde{x})$ binds names of $\tilde{x}$. Agent invocation $A\langle\tilde{y}\rangle$ is then the use of P_A term where all occasions of names from $\tilde{x}$ are substituted by $\tilde{y}$.

The distributed system is composed of places. Place $[P]_p$ is defined by its name p and agent term P which is computed inside the place. System $S_1 \| S_2$ is parallel composition of independent places in S_1 and S_2. Given a system S, we assume the existence of function *sites* which returns the set of places of S. The composition $S_1 \| S_2$ is defined only if $sites(S_1) \cap sites(S_2) = \varnothing$, thus we can consider a system just as a set of disjunctive places.

4 Semantics of the System

Presented semantics describes possible evolution of agents, places and whole distributed system without providing the actual allocation of processes and names. We will define operational semantics of the system in a notion of evaluating of the actions.

4.1 Semantics of Software Agents

The rules of agent semantics describe the evolution of an agent. We present labeled transition $P \xrightarrow{\alpha} P'$ where agent P' is derived from agent P by performing action α. Structural rules of the agent semantics are following:

$$\mathbf{r}_p(x).P \xrightarrow{\mathbf{r}_p(x)} P \tag{A1}$$

$$\mathbf{w}_p(x).P \xrightarrow{\mathbf{w}_p(x)} P \tag{A2}$$

$$\mathbf{m}_p(Q).P \xrightarrow{\mathbf{m}_p(Q)} P \tag{A3}$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \tag{A4}$$

$$\frac{P \xrightarrow{\alpha} P'}{Q + P \xrightarrow{\alpha} P'} \tag{A5}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \tag{A6}$$

$$\frac{P \xrightarrow{\alpha} P'}{Q \mid P \xrightarrow{\alpha} Q \mid P'} \tag{A7}$$

$$\frac{P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P' \quad A(\tilde{x}) \stackrel{def}{=} P}{A\langle \tilde{y} \rangle \xrightarrow{\alpha} P'} \tag{A8}$$

Rules (A1), (A2) and (A3) describe how the actions are evaluated by agents. Rules (A4) and (A5) describe behavior of nondeterministic composition of agents, while rules (A6) and (A7) describe semantics of parallel composition of agents. Last rule (A8) describes invocation of agent named A.

We will use the standard notion $P\{\tilde{y}/\tilde{x}\}$ to indicate the simultaneous of any free occurrence of $x \in \{\tilde{x}\}$ with corresponding $y \in \{\tilde{y}\}$ in P.

4.2 Semantics of Distributed System

Semantics of the distributed system is defined by reduction relation ($\rightarrow$) rules which present basic computational paradigm for agent interactions within the system and evolution of the system. In addition the structural congruence ($\equiv$) is defined for the system semantics. Reduction rules are following:

$$\frac{P \xrightarrow{\mathbf{m}_p(Q)} P'}{[P]_p \rightarrow [P' \mid Q]_p} \tag{S1}$$

$$\frac{P_1 \xrightarrow{\mathbf{m}_{p_2}(Q)} P_1'}{[P_1]_{p_1} \parallel [P_2]_{p_2} \rightarrow [P_1']_{p_1} \parallel [P_2 \mid Q]_{p_2}} \tag{S2}$$

$$\frac{P_1 \xrightarrow{\mathbf{r}_p(x)} P_1' \quad P_2 \xrightarrow{\mathbf{w}_p(y)} P_2'}{[P_1 \mid P_2]_p \rightarrow [P_1' \mid P_2']_p} \tag{S3}$$

$$\frac{P_1 \xrightarrow{\mathbf{r}_{p_1}(x)} P_1' \quad P_2 \xrightarrow{\mathbf{w}_{p_1}(y)} P_2'}{[P_1]_{p_1} \parallel [P_2]_{p_2} \rightarrow [P_1'\{y/x\}]_{p_1} \parallel [P_2']_{p_2}} \tag{S4}$$

$$\frac{[P_1]_p \rightarrow [P_1']_p}{[P_1 \mid P_2]_p \rightarrow [P_1' \mid P_2]_p} \tag{S5}$$

$$\frac{S_1 \rightarrow S_1' \quad sites(S_1) \cap sites(S_2) = \varnothing}{S_1 \parallel S_2 \rightarrow S_1' \parallel S_2} \tag{S6}$$

$$\frac{S \equiv S_1 \quad S_1 \rightarrow S_2 \quad S_2 \equiv S'}{S \rightarrow S'} \tag{S7}$$

Reduction rules clearly distinct between local and remote interactions performed by located agents and provide a formal model to guide the implementation.

Rule (S1) describes movement and evaluation of an agent. Agent P evaluates the agent Q at the same place. Agent Q is running in parallel with agents located at

place p. Rule (S2) is very similar to the rule (S1) while agent P moves the agent Q to another place p_2 where it is evaluated in parallel with existing agents (P_2) there.

Rule (S3) describes synchronous communication between two agents located at the same place. The communication is synchronized when both peers want to interact (read or write) within the same place. It means two communication actions $\mathbf{r}_p(x)$ and $\mathbf{w}_{p'}(y)$ will interact when $p = p'$ and then the name y will be substituted for all occurrences of name x in term followed by $\mathbf{r}_p(x)$ prefix. Rule (S4) is very similar to rule (S3) while communicating agents are located on different places.

Rule (S5) describes asynchronous evolution of subcomponents of the place. It means each site of the system is working autonomously.

How the reduction behaves in presence of operator of parallel composition of places is defined by rule (S6).

The reduction behaves with respect to structural congruence as we can see in rule (S7). Structural congruence is defined following way:

$$S_1 \parallel S_2 \equiv S_2 \parallel S_1 \tag{C1}$$

$$(S_1 \parallel S_2) \parallel S_3 \equiv S_1 \parallel (S_2 \parallel S_3) \tag{C2}$$

The rule (C1) shows the operator $\parallel$ is commutative and rule (C2) shows the operator $\parallel$ is associative.

5 Expressing Mobile Code Applications

According to the classification proposed in [8], we can single out three paradigms, apart from the traditional client-server paradigm, which are largely used to build mobile code applications:

- remote evaluation,
- code on demand and
- mobile agent.

However we think of distributed systems based on mobile agents, our model of communication and mobility can describe all three programming paradigms. Now we will show expression of the three mobile code paradigms and some practical examples of mobile code applications.

5.1 Mobile Code Paradigms

5.1.1 Remote Evaluation

Remote evaluation is performed when a client sends a piece of code to the server and server evaluates the code and client can get the results back from the server.

We define term *Client* that sends a request for remote evaluation to the *Server* 's place s. Request consists of a code P and a name of the *Client* 's place c. Then the *Client* reads the result into the name y and continues as C.

Term *Server* reads the request from his local place s. The received code is stored in name x and the name of *Client* 's place is stored in name p. Then the code in x is evaluated and the result r is sent back to the *Client* 's place. The *Server* is performing an independent work in S.

We define the following terms where the whole system defined by term *System* is a parallel composition of *Server* 's place and *Client* 's place:

$$Client \overset{def}{=} \mathbf{w}_s(P,c).\mathbf{r}_c(y).C$$

$$Server \overset{def}{=} \mathbf{r}_s(x,p).x.\mathbf{w}_p(r) \mid S$$

$$System = [Client]_c \parallel [Server]_s$$

5.1.2 Code on Demand

Code on demand describes the situation where a client wants to perform a code that is presented by the server. Client asks for a code and server sends it to the client where it can be evaluated.

We define term *Client* that sends a request to the *Server* 's place s. The request consists of a name of the *Client* 's place c Then the *Client* reads the code from local place into the name x. Finally the code is evaluated and *Client* continues as C.

Term *Server* reads the request from his local place s. The received name of *Client* 's place is stored in p. Then the *Server* sends a code P to the *Client* 's place. The *Server* is performing an independent work in S.

We define the following terms where the whole system defined by term *System* is a parallel composition of *Server* 's place and *Client* 's place:

$$Client \overset{def}{=} \mathbf{w}_s(c).\mathbf{r}_c(x).x.C$$
$$Server \overset{def}{=} \mathbf{r}_s(p).\mathbf{w}_p(P) \mid S$$
$$System = [Client]_c \parallel [Server]_s$$

5.1.3 Mobile Agent

Mobile agent is a paradigm where an autonomous code (agent) is sent from the client to the server. By autonomous we mean that the client and server do not need to synchronize the agent invocation and the agent is running independently and concurrently within the server's place.

We define an abstraction $Agent(\tilde{x})$ of a mobile agent and term *Client* is moving the agent to the *Server* 's place s.

Term *Server* is performing its independent work S and it is able to receive the agent which is then running in parallel with other *Server* 's actions in its local place s.

We define the following terms where the whole system defined by term *System* is a parallel composition of *Server* 's place and *Client* 's place:

$$Agent(\tilde{x}) \overset{def}{=} P$$
$$Client \overset{def}{=} \mathbf{m}_s(Agent\langle \tilde{z} \rangle).C$$
$$Server \overset{def}{=} S$$
$$System = [Client]_c \parallel [Server]_s$$

5.2 Examples

5.2.1 Remote Procedure Call

This example shows that we are able to model very traditional mobile code application that is performing remote procedure call.

A client sends a request to a server and waits for response. The request consists of procedure name and its real parameters that should be performed by a server and the address of the client's place where to send a result.

Term *Client* sends the request with name of the procedure *Proc*, its real parameters $\tilde{z}$ and the name of the *Client* 's place c to the *Server* 's place s. Term *Server* reads from its local place s the request into the x (name of the

procedure), $\tilde{y}$ (parameters of the procedure) and p (name of the *Client* 's place). Then in parallel it runs the *Server* recursively and continues as procedure stored in x with $\tilde{y}$ parameters. When procedure x is finished the result r is sent back to the *Client* 's place which name is stored in p.

The whole distributed system is defined in term *System* where *Client* 's place and *Server* 's place are computed in parallel:

$$Proc(\tilde{x}) \overset{def}{=} P$$
$$Client \overset{def}{=} \mathbf{w}_s(Proc, \tilde{z}, c).\mathbf{r}_c(y).C$$
$$Server \overset{def}{=} \mathbf{r}_s(x, \tilde{y}, p).(Server \mid x\langle \tilde{y} \rangle.\mathbf{w}_p(r))$$
$$System = [Client]_c \parallel [Server]_s$$

5.2.2 Dynamic Data Gethering

This example shows the model of simple mobile agent system. We define a mobile agent, which travels from place to place and searches for information.

A user defined by term *User* needs additional information on a data represented by name z. *User* launches mobile agent *Seeker* that dynamically travels among nodes looking for result information r in distributed database and stores it in y. If the information is found it is sent back to the *User* otherwise the *Seeker* continues in next place. *User* is waiting for the result and in parallel it continues with other independent work U.

Agent $Seeker(x,h,l)$, where x is searched information, h is home place of the *User* and l is local place of the agent, is reading the data from the local place l. It reads either searching result or the name of the next place where to search. In the first case it sends the result stored in y back to the *User*. In he second case it moves a new instance of the agent to the new place p and ends

The whole system is defined in term *System* where each independent place is sending either result information or the next place for searching:

$$User \overset{def}{=} \mathbf{m}_{p_1}(Seeker\langle z, u, p_1 \rangle).(\mathbf{r}_u(y) \mid U)$$
$$Seeker(x,h,l) \overset{def}{=} \mathbf{r}_l(x,y).\mathbf{w}_h(y) +$$
$$+\mathbf{r}_l(x,p).\mathbf{m}_p(Seeker\langle x, h, p \rangle)$$
$$System = [User]_u \parallel [\mathbf{w}_{p_1}(z,r) + \mathbf{w}_{p_1}(z,p_i)]_{p_1} \parallel \cdots$$
$$\cdots \parallel [\mathbf{w}_{p_n}(z,r) + \mathbf{w}_{p_n}(x,p_i)]_{p_n}$$

Conclusions and Future Work

Modeling rules presented in the paper seem to be very suitable tool for formal description of distributed systems based on agent technology and technology of mobile code. The formal semantics is useful for discussing the design of modeled application and provides guidelines for its implementations in programming languages.

Primitive actions defined in the model present communication and mobility as key interactions for mobile agents. Abstraction of places, their parallel composition and performing interactions within places are very natural for distributed system architectures. These approaches in our model differ from very general π-calculus and ambient calculus [9].

Security properties of distributed system are also very important area and research on presented apparatus continues in this field [10, 11]. For example, presence of typing information [12, 13] within the names can provide privacy and security properties. In addition implementation of spi-calculus [14] primitives can add usage of secure communication protocols to the model.

We also work on multi-agent system platform [15] where mobile agents can work together to solve the common tasks. We use these models to define and to make verification of communication schemes [16, 17] for mobile agents coordination and cooperation within the multi-agent environment.

References

[1] Jennings, N. R., Wooldridge, M. J.: Applications of Intelligent Agents. In N. R. Jennings, M. J. Wooldridge (eds.): Agent Technology: Foundations, Applications, and Markets, Springer, Berlin, Heidelberg, New York, 1998, pp. 3-28

[2] Harrison, C. G., Chess, D. M., Kershenbaum, A.: Mobile Agents: Are They a Good Idea? Technical report, IBM Research Division, T. J. Watson Research Center, March 1995

[3] Baeten, J. C. M., Weijland, W. P.: Process Algebra. Cambridge University Press, Cambridge, New York, Port Chester, Melbourne, Sydney, 1990

[4] Šimoňák, S., Hudák, Š.: Petri Net Semantics for ACP Terms. Vol. 4, No. 1, Košice, 2004, pp. 55-59

[5] Milner, R.: Communication and Concurrency. Prentice Hall, 1989

[6] Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, Part I and II, Information and Computation, 100, 1992, pp. 1-77

[7] Milner, R.: Communicating and Mobile Systems: the -Calculus. Cambridge University Press, Cambridge, New York, Melbourne, 1999

[8] Fuggetta, A., Picco, G. P., Vigna, G.: Understanding Code Mobility. Software Engineering, 24(5), May 1998, pp. 342-361

[9] Cardeli, L., Gordon, A. D.: Mobile ambients. Theoretical Computer Science 240, 2000, pp. 177-213

[10] Tomášek, M.: Specification Issues of Communication and Code Mobility. Acta Electrotechnica et Informatica, Vol. 4, No. 3, pp. 56-60, Košice, 2004

[11] Tomášek, M.: Vyjadrenie dynamiky mobilných programov. PhD. Thesis, Technical University of Košice, Košice, 2004 (in Slovak)

[12] Pierce, B., Sangiorgi, D.: Typing and Subtyping for Mobile Processes. In Proceedings of LICS '93, IEEE Press, 1993

[13] Kobayashi, N., Pierce, B., Turner, D.: Linearity and the -Calculus. In Proceedings of POPL '96, 1996

[14] Adabi, M., Gordon, A. D.: A Calculus for Cryptographic Protocols: the Spi-Calculus. In Proceedings of the Fourth ACM Conference on Computer and Communications Security, ACM Press, April 1997, pp. 36-47

[15] Krokavec, M., Paralič, M., Andričík, M., Tomášek, M.: Mobile Agent Based Distributed Programming. In D. Kocur, A. Filasová, J. Šaliga (eds.): 3rd Internal Scientific Conference of the Faculty of Electrical Engineering and Informatics (3rd ISC'2003), pp. 37-38, Košice, Slovakia, May 28, 2003

[16] Tomášek, M.: Concepts for Mobile Agents Interaction. In M. Jelšina, J. Kollar (eds.): Proceedings of Scientific Conference with International Participation Computer Engeneering and Informatics '99 (CEI '99), Košice - Herľany, Slovakia, October 14-15, 1999, pp. 259-264

[17] Tomášek, M.: Distributed System Based on Technology of Mobile Agents. Acta Electrotechnica et Informatica, Vol. 1, No. 1, Košice, 2001, pp. 55-60