Language for a Distributed System of Mobile Agents

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: Types with behavioral scheme for mobile ambients are suitable for expressing the dynamic properties of mobile code applications, where the main goal is to avoid the ambiguities and possible maliciousness of some standard ambient constructions. We can statically specify and check access rights for the authorization of ambients and threads to communicate and move. We define a language which expresses software agents migration in the space of distributed places. This allows us to understand various aspects of code mobility.

Keywords: code mobility; software agents; type system

1 Introduction

Communication between mobile ambients [1] based on a concurrency paradigm represented by π-calculus [2] is represented by the movement of other ambients of usually shorter life which have their boundaries dissolved by an open action to expose their internal threads performing local communication operations. Such capability of opening an ambient is potentially dangerous [3, 4, 5]. It could be used inadvertently to open and thus destroy the individuality of an object or mobile agent. Remote communication is usually emulated as a movement of such ambients (communication packages) in the hierarchy structure.

We intend to keep the purely local character of communication so that no hidden costs are present in the communication primitives, but without open operation. This solves the problem of the dissolving boundaries of ambients, but disables interactions of threads from separate ambients. We must introduce a new operation move for moving threads between ambients. The idea comes from mobile code programming paradigms [6] where moving threads can express strong mobility mechanism, by which the procedure can (through move operation) suspend its execution on one machine and resume it exactly from the same point

on another (remote) machine. This solves the problem of threads mobility and by moving threads between ambients we can emulate communication between the ambients.

The advantages of our approach are shown in the natural way of encoding the semantics of language for adistributed system of mobile agents. First, we discuss the code mobility for better understanding and then we show how to naturally express objective and subjective mobility implemented in various software applications. Respecting all aspects of code migration paradigms, we are able to propose the language for mobile agents distributed system specification.

2 Revised Calculus of Mobile Ambients

Abstract syntax and operational semantics of our calculus are based on abstract syntax and operational semantics of ambient calculus including our new constructions.

2.1 Abstract Syntax

The abstract syntax of the terms of our calculus is the same as that of mobile ambients except for the absence of open and the presence of the new operation *move* for moving threads between ambients. We allow synchronous output and the asynchronous version is its particular case. The abstract syntax consists of two domains:

M	::=	mobility operations
	n	Name
	$in\ M$	move ambient into M
	$out\ M$	move ambient out of M
	$move\ M$	move thread into M
	$M.M'$	Path
P	::=	Processes
	$\mathbf{0}$	inactive process
	$P \mid P'$	parallel composition
	$!P$	Replication
	$M[P]$	Ambient
	$(\nu n : \mathbf{P}[\mathcal{B}])P$	name restriction
	$M.P$	action of the operation

	$\langle M \rangle.P$	synchronous output
	$(n : \mu).P$	synchronous input

2.2 Operational Semantics

The operational semantics is given by reduction relation along with a structural congruence in the same way as those for mobile ambients.

Each name of the process term can figure either as free:

$$fn(n) = \{n\} \qquad fn(\mathbf{0}) = \varnothing$$
$$fn(in\ M) = fn(M) \qquad fn(P \mid P') = fn(P) \cup fn(P')$$
$$fn(out\ M) = fn(M) \qquad fn(!P) = fn(P)$$
$$fn(move\ M) = fn(M) \qquad fn(M[P]) = fn(M) \cup fn(P)$$
$$fn(M.M') = fn(M) \cup fn(M') \qquad fn((\nu n : \mathbf{P}[\mathcal{B}])P) = fn(P) - \{n\}$$
$$fn(M.P) = fn(M) \cup fn(P)$$
$$fn(\langle M \rangle.P) = fn(M) \cup fn(P)$$
$$fn((n : \mu).P) = fn(P) - \{n\}$$

or bound:

$$bn(n) = \varnothing \qquad bn(\mathbf{0}) = \varnothing$$
$$bn(in\ M) = bn(M) \qquad bn(P \mid P') = bn(P) \cup bn(P')$$
$$bn(out\ M) = bn(M) \qquad bn(!P) = bn(P)$$
$$bn(move\ M) = bn(M) \qquad bn(M[P]) = bn(M) \cup bn(P)$$
$$bn(M.M') = bn(M) \cup bn(M') \qquad bn((\nu n : \mathbf{P}[\mathcal{B}])P) = bn(P) \cup \{n\}$$
$$bn(M.P) = bn(M) \cup bn(P)$$
$$bn(\langle M \rangle.P) = bn(M) \cup bn(P)$$
$$bn((n : \mu).P) = bn(P) \cup \{n\}$$

We write $P\{n \leftarrow M\}$ for a substitution of the capability M for each free occurrences of the name n in the term P. Then similarly for $M\{n \leftarrow M\}$.

Structural congruence is standard for mobile ambients:

- equivalence

$$P \equiv P \qquad \text{(SRefl)}$$
$$P \equiv Q \Rightarrow Q \equiv P \qquad \text{(SSymm)}$$
$$P \equiv Q, Q \equiv R \Rightarrow P \equiv R \qquad \text{(STrans)}$$

- congruence

$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$ (SPar)

$P \equiv Q \Rightarrow !P \equiv !Q$ (SRepl)

$P \equiv Q \Rightarrow M[P] \equiv M[Q]$ (SAmb)

$P \equiv Q \Rightarrow (\nu n : \mathbf{P}[\mathcal{B}])P \equiv (\nu n : \mathbf{P}[\mathcal{B}])Q$ (SRes)

$P \equiv Q \Rightarrow M.P \equiv M.Q$ (SAct)

$P \equiv Q \Rightarrow \langle M \rangle.P \equiv \langle M \rangle.Q$ (SCommOut)

$P \equiv Q \Rightarrow (n : \mu).P \equiv (n : \mu).Q$ (SCommIn)

- sequential composition (associativity)

$(M.M').P \equiv M.M'.P$ (SPath)

- parallel composition (associativity, commutativity and inactivity)

$P \mid Q \equiv Q \mid P$ (SParComm)

$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$ (SParAssoc)

$P \mid \mathbf{0} \equiv P$ (SParNull)

- replication

$!P \equiv P \mid !P$ (SReplPar)

$!\mathbf{0} \equiv \mathbf{0}$ (SReplNull)

- restriction and scope extrusion

$n \neq m \Rightarrow (\nu n : \mathbf{P}[\mathcal{B}])(\nu m : \mathbf{P}[\mathcal{B}'])P \equiv (\nu m : \mathbf{P}[\mathcal{B}'])(\nu n : \mathbf{P}[\mathcal{B}])P$ (SResRes)

$n \notin fn(Q) \Rightarrow (\nu n : \mathbf{P}[\mathcal{B}])P \mid Q \equiv (\nu n : \mathbf{P}[\mathcal{B}])(P \mid Q)$ (SResPar)

$n \neq m \Rightarrow (\nu n : \mathbf{P}[\mathcal{B}])m[P] \equiv m[(\nu n : \mathbf{P}[\mathcal{B}])P]$ (SResAmb)

$(\nu n : \mathbf{P}[\mathcal{B}])\mathbf{0} \equiv \mathbf{0}$ (SResNull)

- garbage collection

$(\nu n : \mathbf{P}[\mathcal{B}])n[\mathbf{0}] \equiv \mathbf{0}$ (SAmbNull)

In addition, we identify processes up to renaming of bound names (α-conversion):

$(\nu n : \mathbf{P}[\mathcal{B}])P = (\nu m : \mathbf{P}[\mathcal{B}])P\{n \leftarrow m\} \quad m \notin fn(P)$ (SAlphaRes)

$(n : \mu)P = (m : \mu)P\{n \leftarrow m\} \quad m \notin fn(P)$ (SAlphaCommIn)

The reduction rules are those for mobile ambients, with the obvious difference consisting in the synchronous output and the missing open operation, and with the new rule for the *move* operation similar to the "migrate" instructions for strong code mobility in software agents:

- basic reductions

$n[in\ m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$ (RIn)

$m[n[out\ m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$ (ROut)

$n[move\ m.P \mid Q] \mid m[R] \rightarrow n[Q] \mid m[P \mid R]$ (RMove)

$(n : \mu).P \mid \langle M \rangle.Q \rightarrow P\{n \leftarrow M\} \mid Q$ (RComm)

- structural reductions

$P \to Q \Rightarrow P \mid R \to Q \mid R$ (RPar)

$P \to Q \Rightarrow n[P] \to n[Q]$ (RAmb)

$P \to Q \Rightarrow (\nu n : \mathbf{P}[\mathcal{B}])P \to (\nu n : \mathbf{P}[\mathcal{B}])Q$ (RRes)

$P' \equiv P, P \to Q, Q \equiv Q' \Rightarrow P' \to Q'$ (RStruct)

3 Type System with Behavioral Scheme

The restriction of the mobility operations is defined by types applying a *behavioral scheme*. The scheme allows for setting up the access rights for traveling of threads and ambients in the ambient hierarchy space of the system.

We define types where we present communication types and message types:

$\kappa ::=$	communication type
$\mid \perp$	no communication
$\mid \mu$	communication of messages of type μ
$\mu ::=$	message type
$\mid \mathbf{P}[\mathcal{B}]$	process with behavioral scheme $\mathcal{B}$
$\mid \mathbf{O}[\mathcal{B} \mapsto \mathcal{B}']$	operation which changes behavioral scheme $\mathcal{B}$ to $\mathcal{B}'$

The behavioral scheme is the structure $\mathcal{B}= (\kappa, Reside, Pass, Move)$ which contains four components:

- κ is the communication type of the ambient's threads.
- $Reside$ is the set of behavioral schemes of other ambients where the ambient can stay.
- $Pass$ is the set of behavioral schemes of other ambients that the ambient can go through, it must be $Pass \subseteq Reside$.
- $Move$ is the set of behavioral schemes of other ambients where the ambient can move its containing thread.

3.1 Typing Rules

The type environment is defined as a set $\Gamma = \{n_1 : \mu_1, \ldots, n_l : \mu_l\}$ where each $n_i : \mu_i$ assigns a unique type μ_i to a name n_i.

The domain of the type environment is defined by:

$$Dom(\varnothing) = \varnothing \qquad Dom(\Gamma, n : \mu) = Dom(\Gamma) \cup \{n\}$$

We define two type formulas for our ambient calculus:

$$\Gamma \vdash M : \mu \qquad \Gamma \vdash P : \mathbf{P}[\mathcal{B}]$$

Typing rules are used to derive type formulas of ambient processes:

$$\frac{n : \mu \in \Gamma}{\Gamma \vdash n : \mu} \quad \text{(TName)}$$

$$\frac{\Gamma \vdash M : \mathbf{P}[\mathcal{B}] \quad \mathcal{B} \in Pass(\mathcal{B}')}{\Gamma \vdash in\ M : \mathbf{O}[\mathcal{B}' \mapsto \mathcal{B}']} \quad \text{(TIn)}$$

$$\frac{\Gamma \vdash M : \mathbf{P}[\mathcal{B}] \quad \mathcal{B} \in Pass(\mathcal{B}') \quad Reside(\mathcal{B}) \subseteq Reside(\mathcal{B}')}{\Gamma \vdash out\ M : \mathbf{O}[\mathcal{B}' \mapsto \mathcal{B}']} \quad \text{(TOut)}$$

$$\frac{\Gamma \vdash M : \mathbf{P}[\mathcal{B}] \quad \mathcal{B} \in Move(\mathcal{B}')}{\Gamma \vdash move\ M : \mathbf{O}[\mathcal{B} \mapsto \mathcal{B}']} \quad \text{(TMove)}$$

$$\frac{\Gamma \vdash M : \mathbf{O}[\mathcal{B}'' \mapsto \mathcal{B}'] \quad \Gamma \vdash M' : \mathbf{O}[\mathcal{B} \mapsto \mathcal{B}'']}{\Gamma \vdash M.M' : \mathbf{O}[\mathcal{B} \mapsto \mathcal{B}']} \quad \text{(TPath)}$$

$$\frac{}{\Gamma \vdash \mathbf{0} : \mathbf{P}[\mathcal{B}]} \quad \text{(TNull)}$$

$$\frac{\Gamma \vdash P : \mathbf{P}[\mathcal{B}] \quad \Gamma \vdash P' : \mathbf{P}[\mathcal{B}]}{\Gamma \vdash P \mid P' : \mathbf{P}[\mathcal{B}]} \quad \text{(TPar)}$$

$$\frac{\Gamma \vdash P : \mathbf{P}[\mathcal{B}]}{\Gamma \vdash !P : \mathbf{P}[\mathcal{B}]} \quad \text{(TRepl)}$$

$$\frac{\Gamma \vdash P : \mathbf{P}[\mathcal{B}] \quad \Gamma \vdash M : \mathbf{P}[\mathcal{B}] \quad \mathcal{B}' \in Reside(\mathcal{B})}{\Gamma \vdash M[P] : \mathbf{P}[\mathcal{B}']} \quad \text{(TAmb)}$$

$$\frac{\Gamma, n : \mathbf{P}[\mathcal{B}'] \vdash P : \mathbf{P}[\mathcal{B}]}{\Gamma \vdash (\nu n : \mathbf{P}[\mathcal{B}'])P : \mathbf{P}[\mathcal{B}]} \quad \text{(TRes)}$$

$$\frac{\Gamma \vdash M : \mathbf{O}[\mathcal{B} \mapsto \mathcal{B}'] \quad \Gamma \vdash P : \mathbf{P}[\mathcal{B}]}{\Gamma \vdash M.P : \mathbf{P}[\mathcal{B}']} \quad \text{(TAct)}$$

$$\frac{\Gamma \vdash P : \mathbf{P}[\mathcal{B}] \quad \Gamma \vdash M : \mu \quad \kappa(\mathcal{B}) = \mu}{\Gamma \vdash \langle M \rangle.P : \mathbf{P}[\mathcal{B}]} \quad \text{(TCommOut)}$$

$$\frac{\Gamma, n : \mu \vdash P : \mathbf{P}[\mathcal{B}] \quad \kappa(\mathcal{B}) = \mu}{\Gamma \vdash (n : \mu).P : \mathbf{P}[\mathcal{B}]} \quad \text{(TCommIn)}$$

We say the process is well-typed when we are able to derive a type formula for it using our typing rules. Well-typed processes respect the communication and mobility restrictions defined in all behavioral schemes of the system. It means such a process has the correct behavior.

4 Discussing Mobility of Software Agents

The new operation *move* with its semantic rule (RMove)

$$n[move\ m.P \mid Q] \mid m[R] \rightarrow n[Q] \mid m[P \mid R]$$

allows us to eliminate remote communication which is usually quite difficult to express. By moving threads among ambient we can move their communication part and return back the results of the communication. For example, the elimination of the remote communication between ambient helps us to encode π-calculus in mobile ambient.

Another interesting aspect of the *move* operation is the possibility to express objective mobility. Distinction between subjective mobility and objective mobility is very important. *Objective mobility* means the migration of the process term managed externally. When we want to move an ambient from one place to another, we can use the operation *move* independently of the inner ambient operations. On the other hand *subjective mobility* is the migration of process term which is managed itself. Using *in* and *out* primitives is the expression of subjective mobility of the ambient. In the theory of mobile ambients we sometimes define objective mobility [7] by primitive $go\,N.M[P]$ with its semantic rules

$$go\ (in\ m.N).n[P] \mid m[Q] \rightarrow m[go\ N.n[P] \mid Q]$$
$$m[go\ (out\ m.N).n[P] \mid Q] \rightarrow go\ N.n[P] \mid m[Q]$$

The *go* operation allows similar movement of the ambient as *in* and *out* where only one ambient boundary is crossed. The *move* operation moves process terms between neighbor ambients, which means crossing two ambient boundaries. This is a possible disadvantage, but it is in the interest of the dangerous *open* primitive avoidance. We decided to adopt this operation because of its importance in the context of software mobility and for its background in the Dπ [8] variant of π-calculus. Another argument is the simplicity and understandability of the type system.

The meaning of objective and subjective mobility we can show in the example of a server for software agents. The mobility of agents is the autonomous process and no external impact is needed. The migration is expressed by a travel plan as a sequence of *in* and *out* operations

$$s_1[...] \mid s_2[a[out\ s_2.in\ s_1.P] \mid ...] \rightarrow s_1[...] \mid s_2[...] \mid a[in\ s_1.P] \rightarrow s_1[a[P] \mid ...] \mid s_2[...]$$

where s_1 and s_2 represent two instances of the server and ambient *a* represents a mobile agent moving between them. In some cases the server can "banish" the agent for various reasons (abusing the system, lack of resources, system overload).

This aspect we can express by objective mobility where the server itself moves the agent to another place

$$s_1[...] \mid s_2[move\ s_1.a[P] \mid ...] \rightarrow s_1[a[P] \mid ...] \mid s_2[...]$$

Table 1
Abstract syntax of the language of mobile agents

τ ::=	**Agent type**
| **A**[]	Agent type without communication
| **A**[τ]	Agent type with communication type τ
System ::=	**Distributed system of places**
| *nothing*	Empty system
| *place p*[*Room*]	Place p with inner *Room*
| *System* | *System*	Composition of places in the system
Room ::=	**Inner of the place**
| *empty*	Empty place
| *agent a* : τ[*Body*]	Agent a of type τ with activity *Body*
| *Room* | *Room*	Compositions of agents in the place
Body ::=	**Agent activity**
| *null*	No activity
| *new a′* : τ[*Body′*].*Body*	Creation of new agent a' of type τ and activity *Body′* on the actual place
| *go p′*.*Body*	Moving agent to place p'
| *read* (m : τ).*Body*	Reading message m of type τ from input
| *write a′*⟨m⟩.*Body*	Writing message m to agent a' on the same place

5 Design of Language for Mobile Agents

Understanding the code mobility and mobility of software agents guide us to define the natural semantics of the mobile applications in the distributed computational environment. We define a language which expresses software agents migration in the space of distributed places. The only operation of agents we consider in this case is the agent communication.

The abstract syntax of the proposed language is in Table 1 together with the informal description of the language constructions. The language semantics is

defined by the encoding to mobile ambient and can be found in Table 2. We can see the encoding follows the dynamical hierarchy of agents and places, which is an advantage of the applied calculus.

Table 2
Denotation semantics of the language of mobile agents

$[\![\mathbf{A}[]]\!] = \mathbf{P}[\mathcal{B}_\perp]$ for $\mathcal{B}_\perp = (\perp, \{\mathcal{B}_{System}, \mathcal{B}_{Room}, \mathcal{B}_\perp\}, \{\mathcal{B}_{Room}, \mathcal{B}_\perp\}, \{\mathcal{B}_\perp\})$
$[\![\mathbf{A}[\tau]]\!] = \mathbf{P}[\mathcal{B}_\tau]$ for $\mathcal{B}_\tau = ([\![\tau]\!], \{\mathcal{B}_{System}, \mathcal{B}_{Room}, \mathcal{B}_\tau\}, \{\mathcal{B}_{Room}, \mathcal{B}_\tau\}, \{\mathcal{B}_\tau\})$
$[\![System]\!] : \mathbf{P}[\mathcal{B}_{System}]$ for $\mathcal{B}_{System} = (\perp, \varnothing, \varnothing, \varnothing)$
$[\![Room]\!]_p : \mathbf{P}[\mathcal{B}_{Room}]$ if $p : \mathbf{P}[\mathcal{B}_{Room}]$ for $\mathcal{B}_{Room} = (\perp, \{\mathcal{B}_{System}\}, \varnothing, \varnothing)$
$[\![Body]\!]_a : [\![\tau]\!]$ if $a : [\![\tau]\!]$
$[\![nothing]\!] = \mathbf{0}$
$[\![place\ p[Room]]\!] = p[[\![Room]\!]_p]$ for $p : \mathbf{P}[\mathcal{B}_{Room}]$
$[\![System \mid System]\!] = [\![System]\!] \mid [\![System]\!]$
$[\![nothing]\!]_p = \mathbf{0}$
$[\![agent\ a : \tau[Body]]\!]_p = (\nu a : [\![\tau]\!])a[[\![Body]\!]_a]$
$[\![Room \mid Room]\!]_p = [\![Room]\!]_p \mid [\![Room]\!]_p$
$[\![null]\!]_a = \mathbf{0}$
$[\![new\ a' : \tau[Body'].Body]\!]_a = (\nu a' : [\![\tau]\!])a'[out\ a.[\![Body']\!]_{a'}] \mid [\![Body]\!]_a$ for $a' \neq a$ and $a : [\![\tau]\!]$
$[\![go\ p'.Body]\!]_a = out\ p.in\ p'.[\![Body]\!]_a$
$[\![read\ (m : \tau)]\!]_a = (m : [\![\tau]\!]).[\![Body]\!]_a$ for $a : [\![\mathbf{A}[\tau]]\!]$
$[\![write\ a'\langle m\rangle.Body]\!]_a = move\ a'.\langle m\rangle \mid [\![Body]\!]_a$ for $m : [\![\tau]\!]$, $a : [\![\mathbf{A}[\tau]]\!]$ and $a' : [\![\mathbf{A}[\tau]]\!]$

Agents define communication type τ in the form of $\mathbf{A}[\tau]$, which expresses that the agent can communicate messages of type τ. A closer look shows us that the agent can communicate only to another agent of the same communication type no matter the direction of the communication. This is given by the possibility of the communication thread movement defined in the *Move* set of the agent's behavioral scheme. We can think of a more general solution, but for better understanding we use this limitation for one behavioral scheme. On the system level there is no communication, so its behavioral scheme defines no communication type. The same is for the distributed places.

Communication between agents takes place in the ambient of the agent accepting the message. We consider only communication of agents located on the same

place. Remote communication we can implement by e.g. complex information about communication place and moving agents there. The message exchange is asynchronous. Synchronous communication is more natural, but its expression is more complex.

Mobility rules are given very simply and statically assuming the places are immobile and agents are moved only through places. For simplicity and better understandability, we consider only one general behavioral scheme for all distributed places in the system. This does not allow us to restrict the movement through the places, but of course we can consider also more complex movement management. To keep the type system correct, we must allow moving agents through agents on the same place, which results from the command *new* from agent creation.

Conclusions

The usage of type system is limited by its very simplicity and it does not prevent more restrictive properties from being checked at runtime. In our related work [9] we proved the soundness theorem for the type system, we demonstrated the system by showing how to model some common mobile code paradigms, we demonstrated some typical mobile code applications and as an expressiveness test, and we showed that well-known π-calculus of concurrency and mobility can be encoded in our calculus in a natural way.

In this work we discussed mobility aspects of software agents and we identified the objective and subjective mobility. Understanding the code mobility was provided by our revised calculus of mobile ambient and types enhanced by behavioral scheme. We were able to propose a very simple language for distributed system of mobile agents. The agents' encoding respects the way of hierarchical distribution of ambients and naturally expresses mobility. The simplicity of the language does not allow us to show more complex constructions, e.g. remote communication and restriction of the movement.

References

[1] Cardelli, L., Gordon, A. D.: Mobile Ambients. Theoretical Computer Science, Vol. 240, No. 1, 2000, pp. 177-213

[2] Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, Part 1 – 2. Information and Computation, Vol. 100, No. 1, 1992, pp. 1-77

[3] Levi, F., Sangiorgi, D.: Controlling Interference in Ambients. Proceedings of POPL'00, ACM Press, New York, 2000, pp. 352-364

[4] Bugliesi, M., Castagna, G.: Secure Safe Ambients. Proceedings of POPL'01, ACM Press, New York, 2001, pp. 222-235

[5] Bugliesi, M., Castagna, G., Crafa, S.: Boxed Ambients. In B. Pierce (ed.): TACS'01, LNCS 2215, Springer Verlag, 2001, pp. 38-63

[6] Fuggeta, A., Picco, G. P., Vigna, G.: Understanding Code Mobility. IEEE Transactions on Software Engineering, Vol. 24, No. 5, May 1998, pp. 342-361

[7] Cardelli, L., Ghelli, G., Gordon, A. D.: Mobility Types for Mobile Ambients. Proceedings of the ICALP'99, LNCS 1644, Springer Verlag, 1999, pp. 230-239

[8] Hennessey, M., Riely, J.: Resource Access Control in Systems of Mobile Agents. Technical Report 2/98, Computer Science Department, University of Sussex, 1998

[9] Tomasek, M.: Expressing Dynamics of Mobile Programs. PhD thesis, Technical university of Košice, 2004