

Multilevel Environments in Insertion Modeling System

Dmitriy M. Klionov¹

¹Kherson State University, 40 rokiv Zhovtnya st. 27, Kherson, Ukraine

soulslayermaster@gmail.com

Abstract. The goal of this paper is to show that the Insertion Modeling System[1] developed by A.A. Letichevsky of the department 100/105 of the Glushkov Institute of Cybernetics, National Academy of Science of Ukraine, Kyiv, Ukraine, can be used as an instrument for the modeling and analysis of complex distributed systems, such as a client-server architectures. The Insertion Modeling [1] is based on the interactions of environments and agents inserted into that environments. Agents have different behaviors represented as Behavior Algebras, and can also be the environments themselves, having another agents with different behaviors inserted into them. The definition for multilevel environments was first given in a paper [1], and was slightly extended in following papers.

Keywords. Insertion modeling, multilevel environments, compatibility relation, client-server architecture

Key terms. Computation, Model, Insertion Modeling

1 Introduction

Insertion modeling is a technology for specification and verification of complex distributed systems based on the interactions of agents and environments. Agents and environments are models of some entities of real world or components of complex systems on different levels of abstraction that interact with one another by means of insertion functions. Also if the environment is considered as an agent it can also be inserted to other environments. In order to model complex systems those consist of a lot of components that have hierarchical structure, the notion of multilevel environments, with agents that are able to move from one environment to another is required. The notion of mobility of such mobile agents are based on the approach recently favored in declarative mobile language design is using mobile calculi that extend or modify the π -calculus [10] with new features, including mechanisms for encryption and security. Calculi of this kind include, among others, the Spi Calculus [6], and the Ambient Calculus [7]. In addition, there is a broader body of work favoring declarative approaches, including work in the field of coordination languages. There has also

been a great expansion of the capabilities and security of agent-based languages such as OAA [10] and D'Agents[13].

According to the Ambient Calculus [7], devised by Luca Cardelli the main difficulty of mobile computations in Web is not in mobility itself but in handling of administrative domains. In the early days of the Internet one could rely on a flat name space given by IP addresses; knowing the IP address of a computer would very likely allow one to talk to that computer in some way. This is no longer the case: firewalls partition the Internet into administrative domains that are isolated from each other except for rigidly controlled pathways. System administrators enforce policies about what can move through firewalls and how.

The client-server model is the prevalent approach in computer networking. The model assigns one of two roles to the computers in a network: a client or a server. A server is a computer system that selectively shares its resources; a client is a computer or computer program that initiates contact with a server in order to make use of a resource. Data, CPUs, printers, and data storage devices are some examples of resources. This model can be represented as a set of administrative domains, with defined access rules, or as some architectural design pattern, like three-tier pattern. Both of these are presented in this paper in terms of the insertion modeling.

2 Insertion Modeling System

Insertion modeling system is an environment for the development of insertion machines and performing experiments with them. Insertion model of a system represent this system as a composition of environment and agents inserted into it, using the insertion function. Contrariwise the whole system as an agent can be inserted into another environment. In this case we speak about the internal and external environment of a system. Agents inserted into the internal environment of a system themselves can be environments with respect to their internal agents. In this case we speak about multilevel structure of agent or environment and about high level and low level environments.

Agent and environments have a set of action and a set of behaviors (processes), defined in behavior algebra. Two set of actions: a set of environment actions and a set of agent actions define the type of environment. If an agent is about to be inserted into the environment at least one of its actions must be allowed by this environment. So the set of agent actions define the type of environments it can be inserted in, as well as the environment's set of allowed agent actions define the type of agents that can be inserted into this environment. Such a relation between types of agents and environments is called compatibility relation [2], which defines a directed graph. When an agent is inserted into some environment, it is able to move to another environment if it is compatible with this environment. For example the rule(1) shows an agent u that moves to an external environment E , from environment R , it is currently inserted into.

$$\frac{u \xrightarrow{\text{move } e} u'}{E[R[u]] \xrightarrow{\text{move_up}(r \rightarrow e)} E[u', R[]]} P(E, R, u, \text{move } e) \quad (1)$$

Here e and r – are the names of environments, $R[]$ – describes environment R that currently have no agents inserted into it. Insertion only occurs if a predicate P is true, and in general case it may depend only on the types of agents and environments. This example rule shows “one step” movement of an agent u , and if the new state of agent u' has the same type as u , and types of environments E and R had not changed as well, rule (1) can be considered as commutative. Also “long range” movements can be defined recursively, for any set of environments between E and R .

3 Insertion Models of Client-Server Architecture

3.1 Domain Model

This model describes a client-server-architecture as a set of administrative domains that have certain access rules. Each of these domains is represented by an environment in IMS. Agents are messages that travel over these domains, trying to access certain protected area of some administrative domain of the server. As an example we take our website `apsystem.or.ua`. It is shown at picture below. The top-most environment E - represents some network (local-area network or internet), with environments of `apsystem` itself, and a set of clients $C1, C2, \dots, Cn$ inserted into the network. Client environments create agents and send them over the network in order to gain access to some function of `apsystem` if they have certain permission, or to a domain of another client. One of the clients can represent a villain (Hacker), which goal is to find all possible security risks and ways of an attack to curtain security protocol.

In order to access administrative domain and to authorize on a server the client has to show that it knows some secret, which is only known to client and server (or two clients that want to exchange some data), and which is not transferred over the network. This key is used to encode messages (transferred by the agents), and when agents tries to move into the environment of administrative domain, this key is used to decode the message, if it is possible than agent inserts into the environment, and proceeds further. There are many ways for generating such secret.

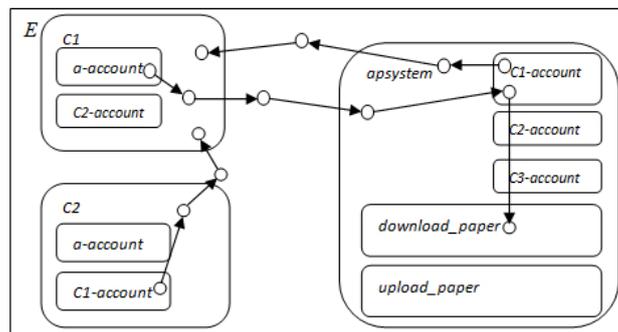


Fig. 1. The domain model of client server

This model uses the standard Needham–Schroeder Public Key protocol [21]. Each client and server has a secret key, which is used to decode messages encoded with appropriate public key. When an agent gets inside the administrative domain (apsystem for example), it have to get a permissions to act inside it. The message transferred by this agent, contains the information about the access rights of the client who sent it. This data is used to move further. When an agent reaches some function (“download_paper” for example) it has a permission to, it is to be sent back by the server to the client. Account environments that are inserted into the clients and the top-most environment of the server store all information required to authorize at appropriate client or server. Tables below show all types of environments and agents.

Types of environments of clients and the top-most environment of the server, are identical. In general the client differs from the server only by the means of environments inside it, which require an action authorized_move.

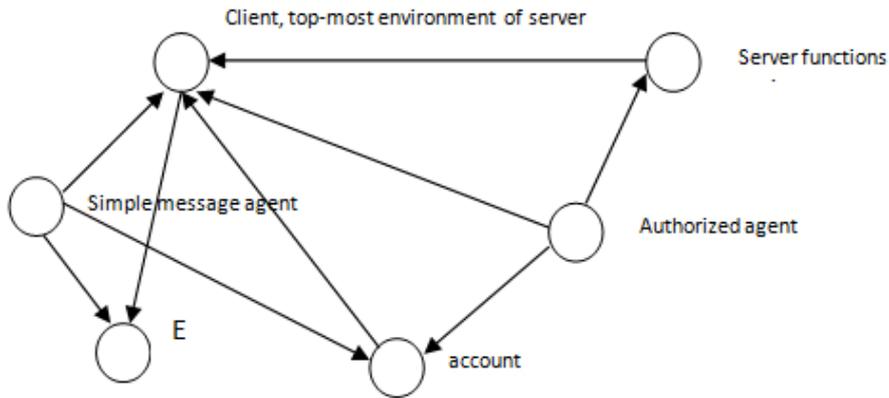


Fig. 2. Compatibility graph for the client-server domain model

Vertexes represent agents and environments, and edges represent a compatibility relation. Directions mean that for example the authorized agent can be inserted into the environments of the account, server functions environments, clients and servers environments.

Interactions with agents:

$$\frac{u \xrightarrow{\text{send } a} u'}{E[C[AC[u]], A[]] \xrightarrow{\text{send } a} E'[C'[AC[]], A'[u']]} P(E, C, AC, A, u, \text{send } a) \quad (2)$$

In equation (2) send(A) Means that client C sends the message u, with an appropriate account AC, to the server A[], over the network E, where a – is the name of server A[]. The definition A[] means that there were no agents inserted into this environment.

Table 1. Actions of agents and environments in domain model

Agent / Environments type	Attributes	Actions
Simple message	<i>mb</i> – message body, actual information carried by this agent;	<i>send a</i> - makes agent to move to the server environment named <i>a</i> ,
	<i>sender</i> – the name of the one who sent this message;	
	<i>enc key</i> – key that is used by encryption algorithm;	<i>access d</i> - agent tries to authorize in order to enter the environment named <i>d</i> , that is in the server environment.
Authorized message	<i>mb</i> – message body, actual information carried by this agent;	<i>auth move d</i> - “authorized move” to some internal environments of the server named <i>d</i>
	<i>role</i> – defines the access level of this information	<i>get_data(x)</i> - agent shares the data it carries.
		<i>invoke(x)</i> - invokes the main function of the environments of the server functions, <i>x</i> – is the access level of authorized agent. It receives as an answer or the result of execution of function, or the “access denied” message.
<i>done(x)</i> - required to check if the result it carries is equivalent to the expected result		
Clients and the top-most environment of the server Allowed actions: send, access, authmove	Secretkey – an integer value of the client’s secret key, that is used by the Needham-Schroeder algorithm Nounce – a place for random numbers.	<i>allow(y)</i> - environment checks the incoming message from the server, <i>y</i> – is the secret key that is used to decode the information from that message.
Accounts Allowed actions: access, authmove, send, get_data(y), done(z)	<i>server</i> – the name of the server it belongs to	<i>update(x)</i> - account is able to update its data about the secret keys used in the Needham-Schroeder algorithm
	<i>role</i> – an integer value that represent a role of this account at server	<i>check_goal(x)</i> - checks if the result brought by the message, is equal to the expected result that is <i>x</i>
	<i>publickey</i> – the public key of the server, that is used by the Needham-Schroeder algorithm	<i>create(r,t)</i> - environment creates agent named <i>r</i> , which has type <i>t</i>
	<i>secret</i> – that will be obtained by Needham-Schroeder algorithm	
Environments that represent server functions(download_paper, upload_paper) Allowed actions: authmove, invoke	<i>permission</i> – an integer value indicating what the required permissions to access it are.	<i>check_permission u</i> - checks if the access level of agent <i>u</i> is appropriate for performing action, if it do then it is delta, if not then agent receives a message that it has no rights to perform the function of this environment
E Allowed actions: send a		

$$\frac{A \xrightarrow{\text{allow}(y)} A', u \xrightarrow{\text{access } ca} u'}{A[u, CA[], D[]] \xrightarrow{\text{access } ca} A[CA[u], D[]]} P(A, CA, u, \text{access } ca) \quad ; \quad (3)$$

An agent u tries to gain access to the server $A[]$, A tries to authorize it, using the secret y , if the authorization succeeds, then u enters appropriate account on the server that is CA , and ca is its name.

$$\frac{CA \xrightarrow{\text{create}(r,t)} CA', u \xrightarrow{\text{get_data}(x)} u'}{A[CA[u], D[]] \xrightarrow{\text{create}(r,t) \ ca} A[CA[u', r], D[]]} P(CA, u, t, \text{get_data}(x), \text{create}(r, t)) \quad (4)$$

An account environment CA creates a new agent named r , which type is t . It carries all data received from u , by the action $\text{get_data}(x)$, x – is that data. This rule creates an agent of type authorized_agent , but it can create an agent of any type that is compatible with this environment.

$$\frac{r \xrightarrow{\text{authmove } d} r'}{A[CA[u, r], D[]] \xrightarrow{\text{authmove } d} A'[CA'[u], D'[r']] } P(A, CA, D, r, \text{authmove } d) \quad (5)$$

The authorized agent u moves to the environment $D[]$, that represent one of the server functions;

$$\frac{D \xrightarrow{\text{check_permission } u} D', r \xrightarrow{\text{invoke}} r'}{D[r] \xrightarrow{\text{invoke}} D'[r']} P(D, r, \text{check_permission } r, \text{invoke}) \quad (6)$$

The agent u invokes the main function of $D[]$, and depending on the result of $\text{check_permission } u$, the result of this invoke might be different.

$$\frac{AC \xrightarrow{\text{check_goal}(x)} AC' r \xrightarrow{\text{done}(y)} \Delta}{AC[r] \longrightarrow AC[\Delta]} P(AC, r, \text{check_goal}(x), \text{done}(y)) \quad (7)$$

When an agent comes back to the client that sent it, the client checks the message it carried, and it matches the required result then it is successful termination. These rules only work if both the client and the server share a secret, known only to them. In order to safely generate such secret the Needham–Schroeder public key algorithm is used. Usually the Needham–Schroeder protocol requires a second server that hosts all the public keys, but for simplicity we assume that all clients and servers know all the public keys. If the secret has already been created, than it is taken instead of public key and secret key for encoding and decoding of messages.

It runs as follows:

1. First we check if the *secret* exists for an account A , if not we send message to the server $A[]$ by the rule (2), and set the value of an agent's attribute mb to N_j that is a simple random number.
2. Then the server $A[]$ uses the rule(3) to decode message using the secret key of server $A[]$,.if the *secret* is not created yet.

3. Then the server replies by the rule (2) to client C the value of mb is set to (N_1, N_2) , N_1 – is the random number created by the client C, and N_2 – is the new random number.
4. If the first part of the mb is equal to the random number that was generated before, than C can take the pair (N_1, N_2) , as a *secret* for the account A.
5. Then C sends a message to A, that contains N_2 . When A will receive it, he will also take the pair (N_1, N_2) , as a *secret* for account C.

In order to verify this protocol one of the clients has to take the role of a villain, its goal is to be authorized as another client from the network, using in this case a man-in-middle attack. [22]

3.2 Insertion Model of Three-Tier Architecture

Unlike of the previous model this one focuses on the actual behavior of data-packages represented by agents, inside the server environment, divided basically to three layers according to the three-tier architecture. The example model of the server hosting two sites apsystem and unarea, is presented.

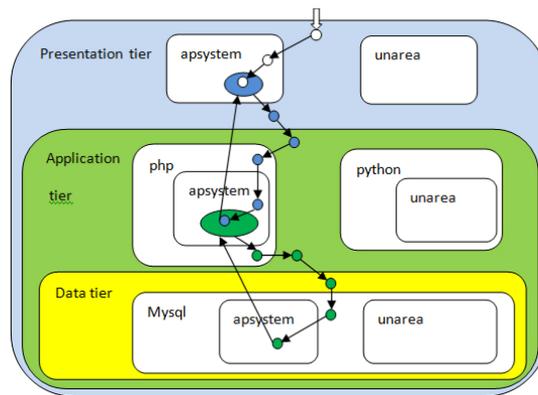


Fig. 3. Insertion model of three-tier client-server architecture

Their frontends are located inside the presentation tier.

$$E[\text{Pr}[\text{aps} [], \text{un} [], [\text{App}[\text{Php}[\text{aps_l}[]], \text{Py}[\text{un_l}[]], [\text{Data}[\text{mysql}[\text{aps_d}[]], \text{un_d}[]]]]]]] \tag{8}$$

(8) is the state of environment in such example. E – the top-most environment, Pr – the presentation tier, aps – the apsystem frontend, un – the unarea frontend, App – the application tier, PHP\ PY – all sites developed in php and python accordingly, aps_l\un_l – the logic of apsystem\unarea, Data – the data tier, aps_d\un_d – the data-base of apsystem\unarea. The user only works with frontend. This means, that the

incoming agent is compatible only with environments of the presentation layer. An agent inserted into one of the frontends carries one request.

Table 2. Types of agents and environments

Agents/Environments types	Actions
User request	execute(x) - executes the request brought by user, x-is the request data User_move d - User agent moves to environment, named d
Script request Allowed actions: execute(x), User_move d	execute_script(y) - executes the request brought by script, y-is the request data Script_move d - script agent moves to environment named d
Data base request Allowed actions: execute_script(y), Script_move d	Execute_query(z) - Executes the request brought by data base, z-is the request data Data_base_move d - Data base agent moves to environment named d
Environments of the Presentation tier Allowed actions: execute(x), User_move d, Script_move d	Create (r,t) - Creates agent named r, of the type t
Environments of the Application tier Allowed actions: execute_script(x), Script_move d, Data_base_move d	Create (r,t) - Creates agent named r, of the type t
Environments of the Data tier Allowed actions: execute_query(x), Script_move d, Data_base_move d	

Interaction with environments: In the rule (9) u gets inside Pr using user_move pr, where pr is the name of the environment Pr, if P can allow this. (a simple one step insertion). The same way u gets inside the environment aps[], using in(aps). This shows how a user goes to some web-site (apsystem in our case), in order to download a page for example. In order to do so he has to load a web-page that has a required link to the paper he wants.

$$\frac{u \xrightarrow{\text{user_move } pr} u'}{E[u, Pr[APS[], App[APS_l[]], Dat[mysq[APS_d[]]]]] \xrightarrow{\text{user_move } pr} P(Pr, u, \text{user_move } pr)} \quad (9)$$

$$E[Pr[u', APS[], App[APS_l[]], Dat[mysq[APS_d[]]]]]$$

The link to the paper is stored in the site's data base that is inside the Data tier, and the rules for extracting these data, and displaying them is inside the Application layer. So, the frontend part (environment of apsystem in our case) creates a new agent, that is compatible only with this environment, and with according environments of the Application layer:

$$\frac{u \xrightarrow{\text{execut}(x)} \Delta}{E[\text{Pr}[APS[u], Q]] \xrightarrow{\text{execut}(x)} E'[\text{Pr}'[APS'[\Delta], Q]]} P(APS, u, \text{execut}(x)) \quad (10)$$

Here we check if u is able to execute its request x if it succeeds than it is DELTA, if it is NOT able to, than we have to check if there any environments inside aps , that are compatible with u , go inside them, and again try the same rule. Q is put for simplicity; it describes all rest of environments that are not involved in the current rule. If there are no such environments or even after insertion to such environment u is still unable to solve(x), then we have to create a new agent r that will get necessary data from application tier.

$$\frac{APS \xrightarrow{\text{create}(r,t)} APS'}{E[\text{Pr}[aps[u], Q]] \xrightarrow{\text{create}(r,t)} E'[\text{Pr}'[aps'[u, r], Q]]} (APS, t, \text{create}(r, t)) \quad (11)$$

Note: r has to be created inside that environment, which u is currently inserted in.

$$\frac{r \xrightarrow{\text{execute_script}(y)} r'}{E[\text{Pr}[APS[u], App[PHP[APS_l[r]]]]] \xrightarrow{\text{solve}(y)} E'[\text{Pr}'[APS[u], App'[PHP[APS_l[r']]]]]} P(APS_l, r, \text{execute_script}(y)) \quad (12)$$

The agent r moves to that environment (APS_l). It should go first to the environment PHP, which is the top-environment of all sites based on PHP, and then moves to the environment APS_l . The rule for its movement is similar to the movement of a user request. The execution of script request is different:

$$\frac{r \xrightarrow{\text{script_move_aps}} r'}{E[\text{Pr}[APS[u], App[PHP[APS_l[r]]]]] \xrightarrow{\text{script_move_aps}} E'[\text{Pr}'[APS[u, r'], App'[PHP[APS_l[r']]]]]} P(PR, App, PHP, APS, APS_l, r, \text{script_move_aps}) \quad (13)$$

If the execution succeeds, than r moves back to the environment, which created it. If not, then the environment APS_l , creates a new agent, which is the query for the data tier. The rules are similar.

4 Conclusions

The client-server model can be considered as a prevalent approach in computer networking, and is one of the best examples of complex distributed systems. Two examples of insertion models of client-server architecture are presented in this paper: the domain model – as a set of administrative domains with pre-defined access rules; and a three-tier architecture - a client-server architecture in which the presentation, the application processing, and the data management functions are logically separated. Both these insertion models with multilevel environments and mobile agents can be extended later for more complicated applications, such as the verification of crypto-

graphic protocols, the problem solving, the constraint propagation, the cognitive architectures.

References

1. Letichevsky, A. A.: Insertion Modeling. *Control Systems and Computers*, 6, 3-14 (2012)
2. Letichevsky, A.: Algebra of Behavior Transformations and its Applications. In: Kudryavtsev, V. B., Rosenberg, I. G. (eds.) *Structural Theory of Automata, Semigroups, and Universal Algebra*. NATO Science Series II. Mathematics, Physics and Chemistry, vol. 207, pp. 241-272, Springer (2005)
3. Baranov, S., Jervis, C., Kotlyarov, V., Letichevsky, A., Weigert, T.: Leveraging UML to Deliver Correct Telecom Applications. In: L. Lavagno, G. Martin, and B. Selic (eds.) *UML for Real: Design of Embedded Real-Time Systems*. Kluwer Academic Publishers, Amsterdam (2003)
4. Letichevsky, A.A., Kapitonova, J., Letichevsky, A. Jr., Volkov, V., Baranov, S., Kotlyarov, V., Weigert, T.: Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications. *Computer Networks*, (47), 662–675 (2005)
5. Kapitonova, J., Letichevsky, A., Volkov, V., Weigert, T.: Validation of Embedded Systems. In: R. Zurawski (ed.) *The Embedded Systems Handbook*. CRC Press, Miami (2005)
6. Abadi, M., Gordon, A.D.: A Calculus for Cryptographic Protocols: the spi Calculus. In: *Proc. 4th ACM Conference on Computer and Communications Security*, pp. 36-47, (1997)
7. Cardelli, L., Gordon, A.: Mobile Ambient. In: Nivat, M. (ed.) *Proc. FoSSaCs'98: Foundations of Software Science and Computational Structures*, LNCS 1378, pp. 140–155, Springer-Verlag (1999)
8. Lange, D.B., Oshima: *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley (1998)
9. Kotz, D., Gray, R.S.: Mobile Agents and the Future of the Internet. *ACM Operating Systems Review* 33(3), 7–13, (1999)
10. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes (Parts I and II). *Information and Computation* 100, pp. 1-77 (1992)
11. Martin, D., Cheyer, A., Morgan, D.: The Open Agent Architecture: A Framework for Building Distributed Software Systems. *Applied Artificial Intelligence*, 12, 91–128 (1999)
12. Gray, R.S., Kotz, D., Cybenko, G., Rus, D.: D'Agents: Security in a Multiple-Language, Mobile-Agents System. In: Vigna G. (ed.) *Mobile Agents and Security*, LNCS 1419, pp. 154–187, Springer-Verlag (1998)
13. Milner, R.: *A Calculus of Communicating Systems*, LNCS 92, Springer-Verlag. (1980)
14. Milner, R.: *Communication and Concurrency*. Prentice Hall, (1989).
15. Milner, R.: *The Polyadic π -Calculus: a Tutorial*. Tech. Rep. ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK (1991)
16. Park, D.: *Concurrency and Automata on Infinite Sequences*. In: LNCS 104. Springer-Verlag, (1981)
17. Roggenbach, M., Majster-Cederbaum, M.: Towards a Unified View of Bisimulation: a Comparative Study. *TCS*, 238, 81–130 (2000)
18. ITU-T. Z.120 Recommendation Z.120 (11/99): *Languages for telecommunications applications – Message Sequence Charts (MSC)* (1999)
19. ITU-T. Z.100 Recommendation Z.100 – *Specification and Description Language (SDL)* (1999)

20. Rutten, J.: Coalgebras and Systems. TCS, 249
21. Needham, R., Schroeder, M.: Using Encryption for Authentication in Large Networks of computers. *Comm. ACM*, 21(12), 993–999 (1978)
22. Lowe, G.: An Attack on the Needham-Schroeder Public Key Authentication Protocol. *Information Processing Letters*, 56(3), 131–136 (1995)
23. Eckerson, W.: Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. *Open Information Systems*. 3(20), 10, 1 (1995)