*Electronic Notes in Theoretical Computer Science 54 (2001)*

*URL:* [*http://www.elsevier.nl/locate/entcs/volume54.html*](http://www.elsevier.nl/locate/entcs/volume54.html) *11 pages*

*Modelling Node Connectivity in Dynamically* Evolving Networks

*Lorenzo Bettini Michele Loreti Rosario Pugliese Dipartimento di Sistemi e Informatica*

*Universita di Firenze*

[*fbettini,loreti,puglieseg@dsi.unifi.it*](mailto:fbettini%2Cloreti%2Cpuglieseg@dsi.unifi.it)

*Abstract*

*Node connectivity is a key aspect of dynamically evolving networks. We address the problem of expressing, and constraining, node connectivity at a linguistic level. We extend the language Klaim (Kernel Language for Agent Interaction and Mobility), that already provides primitives to explicitly con gure the topology of networks, with the notion of clusters of nodes. This feature permits explicitly grouping nodes thus expressing, and constraining, their communication ability: two nodes can in- teract only if they are in the same cluster. Clusters can be used to model many real situations in a natural way and to easily express a number of basic properties of node connectivity, such as, e.g., locality and distribution of nodes, eÆciency, and fault tolerance. However, they do not add expressive power to the language: to some extent, the extended language can be translated into the original one.*

# *1 Introduction*

*Internet provides means and technologies that permit sharing many resources* and services among several computers geographically distributed in a wide- area network. This is a highly dynamic infrastructure that evolves over time. For instance, new nodes can get connected to the network or existing nodes can disconnect. Connections and disconnections can be temporary and unex- pected. In certain situations a temporary connection can be established \on the y" among terminals equipped with wireless devices, and ad-hoc paths to services can be established to enable communication among components. Moreover, the topology of connections can change and nodes can move else- where in the net. Indeed, some nodes can be mobile devices such as, e.g., laptops, PDAs and cellular phones.

*? This work has been partly supported by MURST Projects SALADIN and TOSCA.*

*c 2001 Published by Elsevier Science B. V. Open access under* [*CC BY-NC-ND license.*](http://creativecommons.org/licenses/by-nc-nd/3.0/)

*By exploiting Internet protocols, distributed applications can exchange* data independently of the underlying architectures and operating systems. However, in the context of dynamically evolving networks, assuming that the underlying network will always be available and that there will always be permanent connections among all nodes hosting resources is too strong.

*Node connectivity is a key aspect of dynamically evolving networks. We* can single out three di erent basic forms of connectivity that a node can employ:

*Tethered mode: Wide-area connectivity is available, thus the node is con-* nected to any other node in the net.

*Disconnected mode: The connection is discontinuous but, in the time inter-* vals when it is available, any other node in the net can be reached.

*Untethered mode: There is no wide-area connectivity, but a local-area con-* nectivity may still be available, in which case, the node is connected to any other node in the same connection area.

*Of course, di erent nodes of a network may use di erent forms of connec-* tivity. The rst form of connectivity represents an idealized situation that seldom takes place in practice for all nodes of a wide-area dynamically evolv- ing network. Here are a couple of realistic, but simpli ed, scenarios where users experiment with the remaining two forms of connectivity.

*Disconnected mode is the usual form of connectivity for a user that does* not own a network access. The user works in isolation on his own documents. Some network access points are available, thus, at some time, the user can connect to the Internet, typically via a modem. Connection is required to update user's data in the central repository or to spawn a query to some server. In the last case, mobile agents could be used to manage the query and collect the results that will be delivered to the user when he reconnects.

*Untethered mode is the usual form of connectivity for a mobile user that,* from time to time, enters di erent areas where connectivity is constrained. For instance, the user may connect to computers that are within a rewall, so that the communication among such computers is available, but connection to the rest of the net is disabled. This limitation, however, could not concern all computers; typically, there might be some of them that are allowed to access the whole net and represent the gateway to the external world.

*Of course, if we consider many users that work in cooperation, we can* combine the previous scenarios and obtain more complex situations where some users are always on line, while others connect now and then and must synchronize their work with the remaining ones.

*Recently, new paradigms and languages for programming applications for* dynamic evolving networks have been proposed that are centered around the notions of location awareness and of mobile agents. A few examples of such languages are Telescript [13], Java [1], Ambients [4] and Klaim [5].

*In this paper we address the problem of expressing, and constraining, node*

*connectivity. More speci cally, we want to devise suitable linguistic abstrac-* tions for managing node connectivity. Thus, we extend the language Klaim, that already provides primitives to explicitly con gure the topology of net- works, with the notion of clusters (i.e. groups) of nodes and with appropri- ate primitives to manage clusters. Clusters permit explicitly grouping nodes thus expressing, and constraining, their communication ability: two nodes can interact only if they are in the same cluster. The primitives for cluster management permit dynamic cluster creation and addition/removal of nodes to/from clusters. The notion of cluster turns out to be convenient to model node connectivity in a natural way and permits easily expressing a number of basic properties of node connectivity, such as, e.g., locality and distribution of nodes, eÆciency, and fault tolerance. However, clusters do not add expres- sive power to Klaim: indeed, to some extent, the extended language can be mapped into the original one.

*The rest of the paper is organized as follows (due to lack of space, all* technical details have been omitted). Section 2 informally presents the lan- guage Klaim, while Section 3 introduces the extensions for managing clusters. Section 4, by means of a simple example, illustrates how to use the new fea- tures to model node connectivity. Finally, Section 5 presents a sketch of the translation of the extended language into Klaim and the extensions to the implementation.

# *2 The Klaim language*

*Klaim (Kernel Language for Agent Interaction and Mobility, [5]) is an exper-* imental programming language speci cally designed to program distributed systems composed of several components interacting through multiple tuple spaces and mobile code. It is inspired by the coordination language Linda [8], hence it relies on the concept of tuple space. A tuple space is a multiset of tuples; these are containers of information items (called elds). Klaim distinguishes between actual elds (i.e. expressions, processes, localities, con- stants, identi ers) and formal elds (i.e. variables). Syntactically, a formal

*eld is denoted with !ide, where ide is an identi er. For instance, the sequence*

*(\foo"; \bar"; !P rice) is a tuple with three elds. The rst two elds are string* values while the third one is a formal eld. Similarly, (out(\hello")@l; \foo") is a tuple whose rst eld is a process. Tuples are anonymous and content- addressable; pattern-matching is used to select tuples in a tuple space:

*two tuples match if they have the same number of elds and corresponding*

*elds do match;*

*a formal eld matches any value of the same type, and two actual elds* match only if they are identical (but two formals never match).

*For instance, tuple (\foo"; \bar"; 100 +200) matches with (\foo"; \bar"; !V al).* After matching, the variable of a formal eld gets the value of the matched

*eld: in the previous example, after matching, V al (an integer variable) will* contain the integer value 300.

*In Linda there is only one global shared tuple space; Klaim extends Linda* by handling multiple distributed tuple spaces. Tuple spaces are placed on nodes that are part of a net. Each node contains a single tuple space and processes in execution; a node can be accessed through its address. There are two kinds of addresses:

*Sites are the identi ers through which nodes can be uniquely identi ed* within a net.

*Localities are symbolic names for nodes. A reserved locality, self, can be* used by processes to refer to their execution node.

*Sites have an absolute meaning and can be thought as IP addresses, while* localities have a relative meaning depending on the node where they are in- terpreted and can be thought as aliases for network resources. Localities are associated to sites through allocation environments, represented as partial functions. Each node has its own environment that, in particular, associates self to the site of the node.

*Klaim processes may run concurrently, both at the same node or at dif-* ferent nodes, and can perform the following basic operations over tuple spaces and nodes:

*in(t)@l: evaluates tuple t and looks for a matching tuple t0 in the tuple* space located at l. Whenever the matching tuple t0 is found, it is removed from the tuple space. The corresponding values of t0 are then assigned to the formal elds of t and the operation terminates. If no matching tuple is found, the operation is suspended until one is available.

*read(t)@l: di ers from in(t)@l only because the tuple t0 , selected by* pattern-matching, is not removed from the tuple space located at l.

*out(t)@l: adds the tuple resulting from the evaluation of t to the tuple* space located at l.

*eval(P )@l: spawns process P for execution at node l.*

*newloc(l): creates a new node in the net and binds its site to l. The node* can be considered a \private" node that can be accessed by the other nodes only if the creator communicates the value of variable l, which is the only means to access the fresh node.

*Klaim processes can be built from basic operations by using standard op-* erators borrowed from process algebras [9], such, e.g., action pre xing and parallel composition. Timeouts can also be used for avoiding that processes block due to network latency bandwidth or, when retrieving information, to absence of matching tuples.

*During tuple evaluation, expressions are computed and localities are trans-* lated into sites. Evaluating a process implies substituting it with its clo-

*sure (i.e. the process together with the environment of the node where the* evaluation is taking place). The di erence between operation out(P )@l and eval(P )@l is that out adds the closure of P to the tuple space located at l, while eval sends P , not its closure, for execution at l. Therefore, if node s1 performs an out of P to node s2, when P is executed at s2 , self will actually refer to s1 . This means that static scoping is used for binding localities. On the contrary, if s1 spawns P at s2 with eval, no closure is sent: P will refer to s2 when using self and dynamic scoping is used for binding localities.

*Due to the separation between the concrete and symbolic address of nodes,* allocation environments have the role of restricting sites visibility and, thus, of partially structuring Klaim nets. However, the model underlying Klaim is at, namely nodes cannot enclose other nodes and all nodes are at the same level, therefore it is not completely suitable for modelling structured nets such as, e.g., the Internet. In [3] we enriched Klaim in order to transform the underlying at model into a hierarchical model. In this paper we study an orthogonal issue, namely the connectivity of nodes.

# *3 Clusters*

*In the current Klaim model, a process P , running at node s1, that knows the* site s2 can communicate with the node s2 . From an abstract point of view, nodes s1 and s2 are connected. Now, if P migrates to node s3 then it is still able to communicate with s2. Abstractly, s3 and s2 get connected while s1 and s2 can possibly get disconnected (this mechanism is reminiscent of \link mobility" in the -calculus [10]). In the Internet, however, the knowledge of the address of a remote host is not suÆcient to communicate with it, since there might be no route to the host.

*In this section we enrich the Klaim communication layer by introduc-* ing the notion of cluster, thus such aspects as mobility and disconnec- tion/reconnection of nodes can be easily modelled. A cluster is a collection of nodes and represents a communication medium shared by all the nodes in the cluster. In fact, two nodes can interact only if they belong to the same cluster (the semantics of Klaim operations is modi ed accordingly). Clusters can be used to model locality and distribution of nodes: nodes in the same cluster are \close" to each other. However, the closeness relation does not need to be speci ed thus, e.g., at a high abstraction level, the whole Internet can be mod- elled as a single cluster of nodes. A single node can belong to several clusters; this models the fact that a node can use several communication media and, thus, can be able to tolerate faults of the connection architecture. Whenever a pair of nodes belong to various clusters, anyone of them can be used for an interaction to take place. In our current framework, the choice of which cluster to use is nondeterministic, however one could easily imagine extensions where the choice depends on eÆciency considerations, and, more generally, on quality of service parameters such as, e.g., bandwidth and latency. As we will

*see, clusters cannot be nested (although, of course, the set of nodes of one* cluster could be contained into the set of the other).

*Operations over clusters modify the topology of networks and can be per-* formed only by superprocesses (these are denoted by P). Superprocesses can- not migrate and cannot be used as tuple eld. They are installed at a node when the network is initially con gured or when the node is dynamically cre- ated by using newloc(u; P), which creates a new node with superprocess P. The superprocess of a node can be thought of as an abstraction of that part of a network operating system that lies at the node, while standard Klaim processes are the user programs that can call for execution of system calls. In addition to the standard Klaim operations, superprocesses can also per- form operations newc, add and rm acting over clusters. More speci cally, newc(w):P creates a new, empty cluster that can be referred to via the vari- able w. add(c):P adds the node where it is running to the cluster c, whilst, conversely, rm(c):P removes the node from c. Hence, node mobility is mod- elled in terms of the ability of nodes in entering to and exiting from clusters.

*As an example, let us consider the net N in Figure 3 (a). Initially, nodes* s1, s2 and s3 are in the same cluster c1 . Afterwards, a superprocess at s1 creates a new (empty) cluster c2 by means of newc(w) (after the operation has been executed, w is bound to c2 ). When add(c2 ) is performed at s1, s1 is added to c2 (Figure 3 (b)). Now, s1 belongs to both c1 and c2. Finally, when rm(c1) is performed at s1 , the node is removed from c1, hence all connections among s1 and the nodes in c1 are broken, while processes in s1 are still running (Figure 3 (c)).

c1

**(a)**

s2

s1

s3

c1

s2

c2

s1

s3

**(b)**

s2

s3

c1

c2

**(c)**

s1

*Fig. 1. Klaim with clusters: an example.*

*To end this section, we show how to use Klaim extended with clusters for* modelling the forms of connectivity presented in the introduction.

*Tethered Mode: There is only one cluster and all nodes belong to it. More-* over, a node cannot be removed from the cluster, hence there are no super- processes in the nodes. The standard Klaim model ts in this picture.

*Disconnected Mode: There is only one cluster and all nodes belong to it.*

*However, a node can be removed from the cluster and work in isolation.* Hence, di erently from the tethered mode, there are superprocesses running on nodes, but new clusters cannot be created.

*Untethered Mode: There is not a single cluster, but a set of clusters, each* corresponding to a di erent local connection area. Superprocesses can add/remove nodes to/from clusters and can create new clusters.

# *4 Disconnected mode: an example*

*Let us consider the following scenario where disconnected mode is the form of* connectivity used.

*A software engineer works in isolation. Sometimes, he dials up and gets reconnected to a virtual community server in order to download updates and perform a query in the virtual community. After that, he goes o -line. Later, he reconnects to the virtual community and the results of the search will be dispatched. Moreover, he can establish a communication with one of the other community users. When he has acquired enough information he logs o . Finally, when he nishes his work, he logs on and uploads the artifacts.*

*Let us now consider a possible speci cation in Klaim that also exploits* clusters and operations over them. In the net that models the scenario there is only one cluster, called virtualc, corresponding to the virtual community. In the cluster, there is a node for the server, called S, and one node for each user in the community. A node is on-line when it belongs to the cluster and, conversely, it is o -line when it is removed from the cluster. We just consider superprocesses and Klaim processes running at the engineer's node. We assume that, initially, the engineer's node is outside the cluster.

*Connection to and disconnection from the cluster are handled by two su-* perprocesses, ConnectionSP and DisconnectionSP respectively, that act upon receiving requests issued by means of tuples of the form (operation, cluster ). Notice that disconnection has to be acknowledged so that the requiring pro- cess can be sure that the node has gone o -line, while connection does not, because any non local operation is blocked until the node is not on-line. The code of the two superprocesses is in Figure 2.

*ConnectionSP = while true do*

*in(\connect", !c)@self.*

*add(c) enddo*

*DisconnectionSP = while true do*

*in(\disconnect", !c)@self.*

*rm(c).*

*out(ack)@self enddo*

*Fig. 2. The code of the superprocesses running at the engineer's node.*

*Let us now consider process EngineerProc that model the engineer's activ-* ity. Initially, process EngineerProc calls for a connection to cluster virtualc to take place. Eventually, the node goes on-line and EngineerProc can down- load the wanted information from the server and send the mobile process Search for execution at S. Then, EngineerProc calls for a disconnection from virtualc to take place and, when the node has gone o -line, a stand alone computation is performed. Later, the process requires a reconnection to the community. When the engineer's node goes back into the cluster, process Search can return the results of its search to the engineer by performing tra- ditional out operations at the engineer's node. Such operations have been suspended during disconnection. Before requiring disconnection for working in isolation, EngineerProc establishes a communication with one of the other nodes (s1) by means of operation out(t1 )@s1 and waits for a tuple matching t2 at self. Finally, after reconnection, EngineerProc uploads his work by means of out(tf ). The schema for process EngineerProc is in Figure 3.

*EngineerProc =*

*out(\connect", virtualc)@self. in(t)@S. eval(Search)@S.*

*out(\disconnect", virtualc)@self. in(ack)@self*

*::: Stand alone computation :::*

*out(\connect", virtualc)@self. out(t1)@s1 . in(t2)@self.*

*out(\disconnect", virtualc)@self. in(ack)@self*

*::: Stand alone computation :::*

*out(\connect", virtualc)@self. out(tf )@S.*

*out(\disconnect", virtualc)@self. in(ack)@self Fig. 3. The schema for process EngineerProc.*

# *5 Translation in standard Klaim and implementation*

*We conclude the paper by rst pointing out a sketch of a possible translation* of the extended language into standard Klaim, and then illustrating how the language extensions will be accommodated in the existing Klaim implemen- tation.

*In the translation, every cluster c in the net becomes a node with site sc;* moreover, if a node s belongs to c then the tuple (s) is inserted in the tuple space at sc and the tuple (cluster; sc) is inserted in the tuple space at s. As regards the operations acting over clusters, newc is simply translated into newloc. Operation add(c) performed at s is translated into a sequence of operations that rst add the tuple (s) to the tuple space at sc, then add the tuple (cluster; sc) in the tuple space at s. More precisely, add(c) is rendered as

*out(self)@sc:out(cluster; sc)@self. Similarly, operation rm(c) is translated into the sequence in(self)@sc:in(cluster; sc)@self.*

*Standard Klaim communication and migration operations with the new* semantics illustrated in Section 3 are translated into sequences of operations that rst test for the presence of a cluster that permits the connection and then perform the Klaim operation. Finding a cluster that permits the connection means nding in the local tuple space a tuple of the form (cluster; s) such that the tuple space at s contains the address of the node where the Klaim oper- ation should act. While conceptually simple, in practice this search requires a more sophisticated management of tuples of the form (cluster; s): a counter should also be used for being able to exhaustively examine all the clusters (and, then, their tuple spaces) to which a node belongs. Timeouts are also used to avoid blocking a process while it searches the tuple space of a cluster node.

*It is possible to show that the original semantics can simulate all the com-* putations of the extended semantics. Of course, the converse is not true because the translation does not preserve atomicity (e.g. deadlocks are not preserved too).

*As regards the implementation, clusters can be smoothly accommodated in* the existing Klaim implementation 1 , without making use of the translation sketched above. Indeed, a Klaim net is implemented through a server where nodes must register by using their site. The server allows nodes to communi- cate both directly or indirectly (i.e. messages pass through the server) and it can be considered as a cluster. Hence, extending the implementation consists in having more than one server in the same net and in allowing a node to register in more than one server. Currently, the extended implementation is in progress.

# *6 Related Work and Conclusions*

*The paradigm that is closer to ours is the Ambient calculus [4]. However,* the aim of clusters and their features are quite di erent from ambients' ones. Clusters represent communication capabilities, rather than real physical en- vironments (indeed clusters are more similar to channels). Hence, di erently from ambients, clusters can overlap so that, e.g., shared nodes can act as gate- ways between di erent clusters. Moreover, clusters cannot move and cannot be nested (in the sense that a cluster cannot occur inside another one), while

*1 The implementation of Klaim consists of Klava, a Java package that provides the run-time system for Klaim operations, and X-Klaim [2], a programming language that extends Klaim with a high level syntax for processes: it provides variable declarations, Klaim operations, assignments, conditionals, sequential and iterative process composi- tion. A compiler is also provided, which translates X-Klaim programs into Java programs that use Klava. X-Klaim syntax and software can be found on-line, at the Klaim site (*[*http://music.dsi.unifi.it).*](http://music.dsi.unifi.it/)

*these are the main features of ambients. Similar considerations also hold for* the Seal calculus [12].

*Another related paradigm is Lime [11]. In Lime, each mobile agent has its* own tuple space and all the agents running over an host share the transient tuple space formed by the union of the tuple spaces of each agent. Lime hosts can resemble our clusters, however, in Klaim, clusters can overlap and nodes' tuple spaces are always distinct, thus can be explicitly addressed.

*Clusters have turned to be a powerful abstraction for modelling node con-* nectivity, a key aspect of dynamically evolving networks. Clusters can be used to model many real situations in a natural way and to easily express a number of basic properties of node connectivity, such as, e.g., locality and distribution of nodes, eÆciency, and fault tolerance. We plan to integrate our current framework with routing information as in [3], in order to be able to hi- erarchically structure clusters and to have a ne grain control over the routing process of messages. We also plan to extend our framework in order to be able to bound quality of service parameters to clusters, such as, e.g., bandwidth and latency, and to quantify the performance that the connection architecture can guarantee. Finally, we also want to extend the Klaim access control type system [7,6] in order to be able to restrict operations over clusters.

# *References*

*[1] Arnold, K., J. Gosling and D. Holmes, \The Java Programming Language," Addison-Wesley, 2000, 3rd edition.*

*[2] Bettini, L., R. De Nicola, G. Ferrari and R. Pugliese, Interactive Mobile Agents in X-Klaim, in: P. Ciancarini and R. Tolksdorf, editors, Proc. of the 7th Int. IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE) (1998), pp. 110{115.*

*[3] Bettini, L., M. Loreti and R. Pugliese, Structured Nets in Klaim, in: Proc. of ACM SAC 2000, Special Track on Coordination Models, Languages and Applications (2000).*

*[4] Cardelli, L. and A. Gordon, Mobile ambients, in: Foundations of Software Science and Computation Structures (FoSSaCS'98), number 1378 in LNCS (1998), pp. 140{155.*

*[5] De Nicola, R., G. Ferrari and R. Pugliese, Klaim: a Kernel Language for Agents Interaction and Mobility, IEEE Transactions on Software Engineering*

*24 (1998), pp. 315{330.*

*[6] De Nicola, R., G. Ferrari and R. Pugliese, Types as Speci cations of Access Policies, in: J. Vitek and C. Jensen, editors, Secure Internet Programming: Security Issues for Distributed and Mobile Objects, number 1603 in LNCS (1999), pp. 117{146.*

*[7] De Nicola, R., G. Ferrari, R. Pugliese and B. Venneri, Types for Access Control, Theoretical Computer Science 240 (2000), pp. 215{254.*

*[8] Gelernter, D., Generative Communication in Linda, ACM Transactions on Programming Languages and Systems 7 (1985), pp. 80{112.*

*[9] Milner, R., \Communication and Concurrency," Prentice Hall, 1989.*

*[10] Milner, R., The polyadic -calculus: a tutorial, Technical Report ECS-LFCS- 91-180, Dep. of Comp. Sci., Edinburgh Univ. (1991).*

*[11] Picco, G., A. Murphy and G.-C. Roman, Lime: Linda Meets Mobility, in:*

*D. Garlan, editor, Proc. of the 21st Int. Conference on Software Engineering (ICSE'99) (1999), pp. 368{377.*

*[12] Vitek, J. and G. Castagna, Seal: A Framework for Secure Mobile Computations, in: Internet Programming Languages, number 1686 in LNCS, Springer, 1999 .*

*[13] White, J. E., Telescript Technology: The Foundation for the Electronic Marketplace, White paper, General Magic, Inc., Mountain View, CA (1994).*