Tuple Centres for the Coordination of Internet Agents

Andrea Omicini
LIA - DEIS - Università di Bologna
Viale Risorgimento, 2
aomicini@deis.unibo.it

Franco Zambonelli
DSI - Università di Modena e Reggio Emilia
Via Campi 213b
franco.zambonelli@unimo.it

Abstract

The presents the TuCSoN coordination model for Internet applications based on network-aware (possibly mobile) agents. The model is based on the notion of *tuple centre*, an enhanced tuple space whose behaviour can be extended according to the application needs. Every node of a TuCSoN environment provides its local communication space, made up of a multiplicity of independently-programmable tuple centres. This makes it possible both to embed global system properties into the space of components' interaction, thus enabling flexible cooperation over space and time between agents and permitting to easily face many issues critical to Internet applications, such as heterogeneity and dynamicity of the execution environments.

1 Introduction

The design and development of Internet-based applications call for new paradigms, models, languages and tools, effectively supporting the vision of the Internet as a *globally distributed computing system*, where any kind of distributed information and resources can be globally accessed and elaborated.

Traditional distributed applications are designed as sets of entities assigned to a given execution environment, and cooperating in a network-unaware fashion. However, enforcing transparency in a wide area network that lacks of a centralised control, as in the case of the Internet, is not a viable approach, because distribution is hard to be kept hidden. Then, a more suitable paradigm for the design of Internet applications, is *network-aware computing* [17, 37]: the Internet is modelled as a collection of independent nodes, and applications are made up of network-aware entities (*agents*) which explicitly locate and access remote data and resources. Furthermore, the capability of the agents to proactively migrate on different execution environments (i.e., Internet nodes) during their execution (*agent mobility*) leads to a more efficient and natural design [37, 25]. On the one hand, mobile agents can move locally to the resources they need, without requiring the transfer of possibly large amounts of data nor network connection during the access. On the other hand, if data analysis rather than data transfer is the goal of an application component, moving the component to the data is the most natural design choice. From now on, we will globally refer to network-aware agents, either mobile or not, as to *Internet agents*.

To appear in the 1999 ACM Symposium on Applied Computing (SAC '99) Special Track on Coordination Models, Languages and Applications, February 28 - March 2, 1999, The Menger, San Antonio, Texas, U.S.A.

Since traditional programming frameworks and techniques fall short in this context, several systems and programming environments keep on appearing, aimed at facilitating and supporting the development of widely distributed applications based on Internet agents [26]. However, while these systems address many key issues related to network-awareness and mobility [17], they do not provide suitable mechanisms and abstractions for agent coordination.

Most of the proposals rely on some forms of peer-to-peer communication, which we argue not to be adequate to Internet applications, since *(i)* direct communication between widely distributed, asynchronous and independent entities is inherently difficult, *(ii)* direct communication is not expressive enough, *per se*, to deal with issues like heterogeneity, dynamicity and security, and requires the interposition of global and not easily scalable middleware, such as an ORB [30]. To our opinion, instead, the choice and exploitation of a suitable *coordination model* [20] is a key point in the design and development of network-aware applications based on mobile agents. Tuple-based coordination models based on associative blackboards (like Linda [18]) seems to cope well with the Internet scenario, since they enable both spatial and temporal uncoupling in interaction, a feature which is essential in case of widely distributed and mobile entities. Still, tuple-based models are typically data-oriented [32] and lack the flexibility (typical of control-oriented coordination models) required by the intrinsic dynamicity of the Internet scenario: both agent-to-agent communication and agent access to local data are tied to the built-in associative mechanisms integrated in tuple spaces. Thus, any coordination policy not directly supported by the model (like atomically reading two tuples in Linda, for instance) has typically to be charged upon agents, that have to be made aware of the coordination laws. This increases the complexity of agent design, by breaking the logical separation between coordination and algorithmic issues.

On this basis, this paper presents the TuCSoN (Tuple Centres Spread over Networks) model for the coordination of Internet agents, based on a notion of programmable coordination medium, called *tuple centre*. A tuple centre is a tuple space enhanced in that its behaviour in response to agent's triggered communication events can be extended so as to embody application-specific coordination laws. This makes it possible to embed global system properties into the interaction space, by charging tuple centre behaviours of many issues critical to Internet applications, such as heterogeneity and dynamicity of the hosting environments, as well as mobile agent cooperation over space and time.

The paper is organised as follows. Section 2 discusses the issue of coordination for Internet agents. Section 3 introduces the TuCSoN model. Section 4 singles out some viable application areas for TuCSoN-coordinated Internet applications and discusses the impact of TuCSoN on them.

2 Coordination of Internet Agents

In agent-based Internet applications, both agent-to-agent and agent-to-execution environment interactions have to be modelled and managed. As argued in [5], the choice of the coordination model actually represent a key-point in the design process. In spite of that, among the several systems and programming environment aimed at supporting and facilitating the design and development of Internet applications, only a few focuses on the definition of suitable coordination model.

2.1 Direct Interaction Models

In the context of mobile agent systems, direct (peer-to-peer) communication is usually adopted as the main interaction protocol between interacting components. Agent-TCL [22] provides for low-level message-passing primitives, while ARA [33] and MOLE [3] integrate both low-level message-passing and rendezvous-like interaction models. Java-based mobile agents systems [26] adopt the client-server pattern typical of object-based systems, and can also exploit the TCP/IP protocol at the socket level. In all the above cases, interaction with the local resources of and execution environment occur in a client-server way, via a specialised server.

In the context of intelligent multi-agent systems, the issues of inter-agent communication has been widely studied in the past few years [16, 21], leading to the definition of sophisticated interaction models for knowledge exchange that, again, are based on peer-to-peer communication.

To our opinion, these approaches do not generally suit mobile agents and Internet applications. In fact, direct coordination implies a strict coupling between interacting entities in terms of name (who the partners are), place (where the partners are) and time (when to interact). This makes it difficult to deal with both agent mobility and execution environment uncertainty and dynamicity. Current researches in the area of intelligent Internet mobile agents for distributed information search mainly focuses on dynamic decision planning to deal with Internet dynamicity and uncertain knowledge, but still neglect the issue of agent mobility [29].

In general, mobility and the wideness of the Internet make direct inter-agent coordination not scalable: while it is possible to let mobile agents interact in a small sized network by tracing their movements, applying this method to the Internet would be expensive, ineffective and unreliable. In addition, direct inter-agent communication makes it difficult to enforce security policies, given that interaction typically occurs with no control by the hosting execution environments.

The interposition of specialised middleware – as in *CORBA* [30] and in *KQML* [16] – can partially solve the problems of direct communication. However, apart from the uneasiness to scale to a world-wide area, middleware cannot help models based on direct communication in shaping the components' space of interaction according to the application needs. In fact, they do not provide the abstractions and metaphors needed to handle and rule the interaction space, which is what a *coordination model* is actually meant to do.

2.2 Blackboard-based Coordination

Following the classification introduced in [32], *data-driven* coordination models focus on communication information, by making coordination rules depend on information exchanged in the interactions, typically occurring through shared data spaces, i.e., blackboards [15, 18].

Blackboard-based coordination models, by uncoupling interacting entities, intrinsically solve many problems related to mobility and dynamicity. In addition, this approach intrinsically provides better support for security models, since the communication abstractions (where all interactions occur) can easily be fully monitored. Furthermore, the well-known scalability limit of computational models based on shared information spaces – that would made them not applicable at the Internet area – can be easily overcome by enforcing locality: *local blackboards* can be associated to each node, to be used by agents to interact locally to their current execution environment. Finally, models based on *associative* blackboards, like Linda tuple spaces [20], are particularly suitable to data-oriented applications, since associative access permits agents to retrieve data from blackboards through some data-matching mechanism (like pattern-matching, or unification) integrated within the blackboards. This makes it easier to deal with data incompleteness as well as with dynamicity and heterogeneity, which are typical of Internet nodes, when seen as information sources.

The interest in blackboard-based coordination models and their recognised effectiveness has led, in the past, to several proposals that define and implement globally shared information spaces [15, 18, 35]. More recently, blackboard-based models have been proposed for adoption in open distributed systems and applications. Among the others, ActorSpace [1] and JavaSpace [27] adopt an object-oriented blackboard models, to be used in the context of distributed object-oriented applications as a mean to store and associatively retrieve object references. However, neither ActorSpace nor JavaSpace explicitly encourage a model based on local dataspaces associated to the execution environments. The ffMAIN agent system [14] defines mobile agents that interact (both with other agents and with the local resources of the hosting execution environment) via information spaces associated to execution environments and accessed through the HTTP protocol. Ambit [7], a recently proposed formal model for mobile computations, introduces the concept of anonymous coordination: a mobile entity, while roaming through the network, can "attach" a message to a given system, like a post-it, which another entity can later read/retrieve. Neither Ambit nor ffMAIN exploit associative access to information. Instead, Linda-like associative interaction spaces are adopted by the PageSpace coordination architecture for distributed interactive Web applications [10], which can also support the coordination of Java mobile agents.

2.3 Toward Programmable Communication Abstractions

Though the benefits of uncoupled and associative coordination have made Linda a general coordination model for parallel and distributed applications [20], their adoption in the context of Internet and mobile agent applications is still limited to a few proposals. To our opinion, this may be due to the fact that Linda-like coordination *lacks* flexibility and control: both agent-to-agent coordination and access to local data are bound by the built-in data-access mechanisms provided by the blackboard. Then, any coordination policy not directly supported by the model has typically to be charged upon agents. In an open, heterogeneous and unreliable environment as the Internet, this is likely to notably increase the agent complexity, which are forced both to implement in their code the peculiar coordination protocols required by the application and to solve Internet related issues, such as heterogeneity and dynamicity of the information sources on the execution environments. In addition, this makes coordination rules distributed between blackboards and the agents, thus affecting the global application design and breaking the logical separation between coordination and algorithmic issues.

In order to address this general limit of the Linda model, which exhibits itself also in contexts different than Internet applications, proposals have been made to enhance the Linda communication kernel either *(i)* by adding new general purpose primitives, or *(ii)* by making it programmable.

The former approach has been adopted by several researchers in order to face coordination problems which could not be easily solved by means of standard Linda operations: among the other, [35] addresses all the coordination problems which can be reduced to the *multiple read problem* by defining the `copy_collect` primitive. In the area of Internet applications, T Spaces [23] defines shared Linda-like tuple spaces, to be used for the coordination of distributed - possibly mobile - application entities, which enables the dynamic addition of new operations into the tuple space. However, we feel that adding new operations to a shared information space tends to limit the uncoupling of the coordination model, because the application components must be made aware of the admissible operation to effectively interact.

The latter approach, without adding new admissible operations, makes it possible to program the behaviour of the communication abstractions in response to communication events. Because this do not modify the tuple space interface, it has to be preferred to the former approach, in particular in the area of Internet agents, where uncoupling is a strict requirement. However, proposals adopting this approach are limited to the area of closed (non Internet) multi-agent systems. Law-governed Linda [28] makes it possible to super-impose new coordination laws by means of local controllers managing the communication between the agents and the tuple space. $\mathcal{ACLT}$ [31] exploits (multiple) logic tuple spaces as programmable communication abstractions [11] whose observational behaviour, unlike Linda tuple spaces, is not fixed once and for all by the coordination model, but can be tailored to specific application needs.

TuCSoN starts from the above considerations and defines a new coordination model suited to Internet-agents, partially inspired by $\mathcal{ACLT}$ for what relates to the notion of programmable communication abstractions, new with regard to its Internet-oriented organisation. To the best of our knowledge, the only research proposals that defines an Internet-oriented coordination model based on programmable tuple spaces is MARS [6], developed in the context of an affiliated research project, and differing from TuCSoN in both its object-oriented tuple space and its imperative programming style for the specification of tuple space behaviours. Instead, previously described proposals, such as PageSpace and JavaSpace, enables only a limited form of control over the tuple space internal activities, and do not generally permit the definition of new behaviours in response to communication events.

3 The TuCSoN Coordination Model

TuCSoN (Tuple Centres Spread over Networks) is a coordination model for Internet applications based on network-aware and possibly mobile agents. TuCSoN mainly focuses on the coordination of the agent's interaction space, i.e., the interaction with both the execution environment's resources and the other agents, and do not aim to implement a new agent system. Instead, TuCSoN is intended to be associated to an existing agent infrastructure [25], and it assumes that the basic functionalities for agent authentication, execution and migration are already provided. For this reason, the paper disregard the issues strictly related to the agent infrastructure and restricts the focus to the TuCSoN's most relevant features, i.e.:

- the TuCSoN *coordination space*, with its twofold interpretation as either a *global interaction space* made up of uniquely denoted communication abstractions, or as a collection of *local interaction spaces*, and
- the TuCSoN *communication abstractions*, whose behaviour can be defined so as to embed global coordination laws.

The first feature effectively supports the twofold role of Internet agents, as network-aware entities that locate and access Internet data and resources, and as roaming entities that transfer their execution on a site and there interact with the local resources. The second one actually provides the coordination model, which is basically data-oriented, with the flexibility and control required to deal with the complexity of Internet applications.

3.1 The TuCSoN Coordination Space

TuCSoN's *coordination space* relies on a multiplicity of independent communication abstractions, called *tuple centres*, spread over Internet nodes, and used by agents to interact with other agents as well as with the local execution framework. Each tuple centre is associated to a node and is denoted by a locally unique identifier. As shown by the example of Figure 1, each node provides its own version of the TuCSoN *communication space* (that is, the set of the admissible tuple centre's identifiers), by virtually implementing each tuple centre as an Internet service.

As a result, each tuple centre can be identified via either its full Internet (*absolute*) name or its local (*relative*) name. More precisely, tuple centre *tc* provided by the Internet node *node* can be referred to by means of its absolute name *tc@node* from everywhere in the Internet, and by means of its relative name *tc* in the context of node *node*. According to the example of Figure 1, the name `books@cslibrary.tucson.edu` univocally denotes the tuple centre `books` provided by the Internet node `cslibrary.tucson.edu`, while the name `books` may denote one of the three versions provided by the three nodes, according to which is the node where the name is used.

Correspondingly, the TuCSoN coordination space can be seen as providing either a *global interaction space*, featuring a collection of uniquely denoted tuple centres (when referring to absolute tuple centre's names), or a collection of *local interaction spaces*, replicating the same set of identifiers (when referring to relative tuple centre's names).

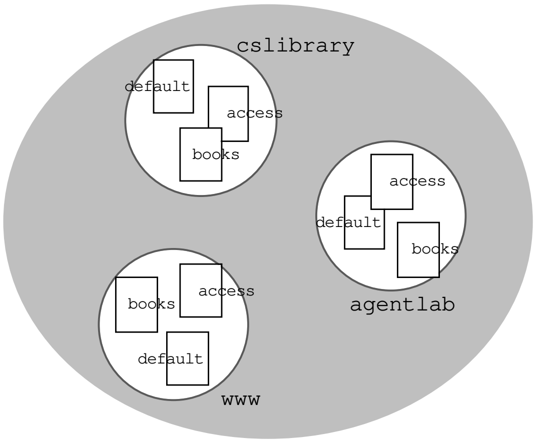

Figure 1: The communication space {`default, access, books`} provided by three TuCSoN nodes {`www, cslibrary, agentlab`}`.tucson.edu`

How this can benefit mobile agent coordination, by simplifying the agent interaction protocol, will be made clearer in the following subsection.

3.2 The TuCSoN Coordination Language

Agents interact by exchanging tuples via tuple centres, by means of a small set of communication primitives (`out`, `in`, `rd`, `inp`, `rdp`) having basically the same semantics of Linda's ones. According to [8], `out` writes a tuple in a tuple centre, while `in` and `rd` send a tuple template and expect the tuple centre to return a tuple matching the template, respectively either deleting it or not from the tuple centre. The primitives `inp` and `rdp` work analogously to `in` and `rd`: however, while the latter wait until a matching tuple can be retrieved from the tuple centre, the former immediately fail if no such a tuple is found.

The general form for any admissible TuCSoN communication operation performed by an agent is

$$tcname?operation(tuple)$$

asking tuple centre *tcname* to perform *operation* using *tuple*. Since *tcname* can be either an absolute or a relative tuple centre's name, agents can adopt two different forms of primitive invocation, the *network* and the *local* one, respectively, according to their contingent needs. The network communication form $tc@node?operation(tuple)$ is used by Internet agents when they behave as network-aware entities, by denoting tuple centres through their absolute names in the global TuCSoN interaction space. For example, this form can be used by a mobile agent to remotely access a candidate hosting environment, query it, and possibly authenticate itself before migrating there. The local communication form $tc?operation(tuple)$, which refers by definition to the local tuple centre's implementation of the current execution node of an agent, is used by Internet agents when they behave as local components of their current hosting environment. This form is typically used by agents to interact with local resources and to exchange data and knowledge with other agents.

The local communication form is not, *per se*, necessary: an agent could always refer to a tuple centre via its absolute name, even when accessing the tuple centres of its current execution environment. However, the availability of both communication forms enforces the separation between network-related issues (such as agent migration across the nodes) and local computation issues (such as the interaction with the resources local to a node), thus reducing agent complexity. The above approach is somehow analogous to the one provided by most file systems, where pathnames can be specified in a relative – context dependent – form, to facilitate file management activities.

The TuCSoN coordination language is not tied to any specific computation language. As in Linda, the semantics of TuCSoN communication primitives has been defined to be exploited by agents written in whatever language, let it be C, Java to Tcl, once the required simple syntax adjustments have been made.

3.3 The TuCSoN Communication Abstractions

As discussed in Section 2, a typical problem of Linda-like coordinated systems relies on the tuple space's built-in and fixed behaviour: neither new primitives can be added, nor can new behaviours be defined in response to communication operations. As a result, either the support provided by the communication device is enough for all needs, or the interacting entities have to be charged in their code of the burden of the coordination.

TuCSoN communication abstractions are tuple spaces enhanced with a notion of *behaviour specification*, in that the behaviour in response to communication events of every tuple centre can be defined according to specific needs. In particular, TuCSoN tuple centres *extends* tuple spaces: an empty behaviour specification for a tuple centre makes it behave exactly as a standard tuple space; a behaviour specification can enclose any needed computational activity. On the one hand, application-specific behaviours can be specified to define to specialised coordination laws for an application. On the one hand, specific behaviours can be defined by the administrator of one node for the local tuple centre in order to define specific semantics for the accesses to its tuple centres performed by agents.

Each interaction space provided by a TuCSoN node relies on a multiplicity of tuple centres. Then, TuCSoN shares the advantages of models [19, 9] based on multiple tuple spaces (such as the enhanced expressiveness of communication, and a more effective support for modularity, information hiding and security), and goes beyond. In fact, the behaviour of every single tuple centre can be defined separately and independently of any other tuple centre according to specific coordination tasks. Thus, coordination laws can be encapsulated into different communication abstractions, providing system designers with a finer granularity for the implementation of global coordination policies.

Generally speaking, the behaviour of a stateful communication abstraction like a shared data space is naturally defined as the observable state transition following a communication event. As a result, defining a new behaviour for a tuple centre basically amounts to specifying a new state transition in response to a standard communication event. This is achieved by enabling the definition of *reactions* to communication events by means of a *reaction specification language* [12]. More precisely, a reaction specification language makes it possible to associate any of the TuCSoN basic communication primitives (`out`, `in`, `rd`, `inp`, `rdp`) to specific computational activities, called reactions. A *reaction* is defined as a set of non-blocking operations, with a success/failure semantics: successful reactions may atomically produce effects on the tuple centre state, failed reactions yield no result at all. Each reaction can freely access and modify the information collected in form of tuples in a tuple centre, and can access all the information related to the triggering communication event (such as the performing agent, the operation required, the tuple involved, ...). Then, the behaviour of a tuple centre can be made as complex as it is needed by the global system requirements: a transition is no longer bounded to be the simple one (such as adding/deleting one single tuple) determined once for all by the model, but can instead be made as complex as desired by the system designer (such as adding/deleting set of tuples, dynamically producing tuples in the tuple centre, and so on).

Each communication event may in principle trigger a multiplicity of reactions: however, all the reactions executed as a consequence of a single communication event are all carried out in a single transition of the tuple centre state. As a consequence, from the agents' viewpoint, the result of the invocation of a communication primitive, which is the sum of the effects of the primitive itself and of all the reactions it triggered, is perceived altogether as a single communication event. For this reason, and because the TuCSoN transactional semantics applies to single communication events, the uncoupling property of the tuple space model is preserved.

3.4 An Example

In order to help understanding the model and some of its features, in this subsection we discuss a simple example of a TuCSoN-coordinated mobile agent application.

Let us consider an application where mobile agents roam through the Internet nodes of a University spread over a large geographic area. We take for instance the case of mobile agents looking for books, which belongs either to some centralised libraries (the general library, the department ones), or to some research groups, and whose information is recorded accordingly in a distributed way. Say, for example, that `cslibrary` is the name of the node of the main Computer Science Department Library, while `agentlab` is the node of the research group on mobile agent systems. Both machines are TuCSoN nodes, and implement the same communication name space as an Internet service. In particular, both provide their (local) version of the tuple centre `books` for accessing information about books which are locally available.

Since `agentlab` contains information about a small collection of books (say, some hundreds), it can reasonably record books directly through tuples in tuple centres. In particular, a tuple `bookTitle(`*`Key`*`,`*`Title`*`)` in the tuple centre `books` of the `agentlab` local communication space records with key *`Key`* a book of the multi-agent group whose title is *`Title`*, while tuple `bookAuthor(`*`Key`*`,`*`Auth`*`)` states that *`Auth`* is (one of) the author(s) of the book whose key is *`Key`*. Instead, since it contains information about a huge amount of books, `cslibrary` exploits a DBMS application where the books of the CS Department are recorded. The DBMS interfaces with the local tuple centre `books` through a wrapper, translating tuples into queries, and back, answers into tuples. The wrapper waits for tuples of the form `query(`*`Query`*`)` and translates them into queries for the DBMS. Then, it waits for the DBMS answer and translates it into a tuple of the form `answer(`*`Query`*`,`*`TableList`*`)`, where *`TableList`* is the answer table provided in form of list. In particular, suppose that a query of the form `query(booksBy(`*`Auth`*`))` makes the wrapper ask the DBMS to return the titles of all

```
...
mySelf = "Curious Agent"
author = "Alan Turing";
while(head(libNode = libraryNodeList)) {
    // ask access to the libNode host
    // with the network communication form
    if (access@libNode?rdp(authorised(id(mySelf)))) {
        // if access is granted move to host lib
        moveTo(libNode);
        // locally access the tuple centre library
        // of the hosting node
        // with the local communication form
        books?rd(authorBooks(author, ?bookList);
        elaborateAndRecord(bookList);
    }
}
...
```

Figure 2: A simple interaction protocol for a TuCSoN mobile agent

the books contained in the database written by *`Auth`*, and the consequent answer is then translated by the wrapper and put into `books` as a tuple `answer(booksBy(`*`Auth`*`),`*`Books`*`)`, where *`Books`* is the list of the books written by *`Auth`* available from the CS Department Library.

Despite of the different architectures of the two sites, mobile agents roaming through the nodes can be designed around a simple and uniform protocol, as shown by the pseudo-code in Figure 2. Consider the mobile agents looking for all the books written by author `"Alan Turing"` available at the University. The agent interacts remotely (as a network-aware entity) with the nodes it needs to explore, using the network communication form, then moves there (if allowed), finally interacts with the host's resources through the local communication space using the local communication form. What is worth to be noticed from Figure 2 is that the interaction with local resources is always performed in the same way independently of the current hosting node: wherever the agent is currently located, it always asks (the local version of) tuple centre `books` for a tuple `authorBooks("Alan Turing",BookList)` through a `rd` operation, expecting that `BookList` is instantiated to the list of the books by `"Alan Turing"` which are locally available. This is achieved by suitably programming the behaviour of tuple centre `books`, then acting as a bridge between the simple and uniform agent interaction protocol and the peculiar structure of each site.

On server `agentlab`, the `books` tuple centre in programmed to react to a communication event of the form `authorBooks(Author,BookLi` by: *(i)* checking whether the required tuple is available; *(ii)* if not, by finding all the tuples `bookAuthor(Key,Author)` and, on the basis of the `Key` field, all the related tuples `bookTitle(Key,Title)`; *(iii)* by merging the above two kinds of tuples in tuples having the form required by agents, i.e., `authorBooks(Author,BookList)`. Server `cslibrary`, instead, is programmed to react to the above communication event by: *(i)* emitting a tuple of the form `query(booksBy(Author))`, which is consumed by the wrapper; *(ii)* when the wrapper return the answer from the DBMS, in terms of a list of titles `BookList`, by producing the required tuple for the agent, i.e., `authorBooks(Author,BookList)`.

The TuCSoN abstract model for the coordination of mobile agents prescinds from the language adopted for tuple centre's behaviour specification, and also from both the kind of the tuples used for the communication and the tuple

```
% books@agentlab behaviour specification
reaction(rd(authorBooks(A,_)), ( pre, no_r(authorBooks(A,_)),
% when attempting to read a tuple authorBooks(A,_)
% which is not avaiable
    out_r(authorKey(A,[])) )).
reaction(out_r(authorKey(A,Keys)), in_r(authorKey(A,Keys)) ).
% for any authorKey(A,Keys) tuple
reaction(out_r(authorKey(A,Keys)), ( in_r(bookAuthor(Key,A)),
    out_r(authorKey(A,[Key|Keys])) )).
add Key to a list of Keys in the tuple authorKey(A,Keys)
reaction(out_r(authorKey(A,Keys)), ( no_r(bookAuthor(_,A)), .
% when there are no more authorKey(A,Keys) tuples
    out_r(keyTitles(A,Keys,[])) )).
    % produce a tuple keyTitles(A,Keys,[])
reaction(out_r(keyTitles(A,Keys,Titles)),
    in_r(keyTitles(A,Keys,Titles))).
reaction(out_r(keyTitles(A,[Key|Keys],Titles)), (
    out_r(bookAuthor(Key,A)), rd_r(bookTitle(Key,Title)),
        out_r(keyTitles(A,Keys,[Title|Titles])) )).
reaction(out_r(keyTitles(A,[],Titles)),
    out_r(authorBooks(A,Titles))).
    % translate it into the required authorBooks(A,Titles) tuple

% books@cslibrary behaviour specification
reaction(rd(authorBooks(A,_)), ( pre, no_r(authorBooks(A,_)),
% when attempting to read a tuple authorBooks(A,_)
% which is not
    out_r(query(booksBy(A))) )).
    % translate it into a query to the DBMS
reaction(out(answer(query(booksBy(A)),Titles)), (
% get the answer tuple from the DBMS
    in_r(answer(query(booksBy(A)),Titles)),
        out_r(authorBooks(A,Titles)) )).
        % translate it into the authorBooks(A,Titles) tuple
```

Figure 3: ReSpecT's behaviour specification for tuple centres `books` provided by nodes `agentlab` and `cslibrary`

matching criterion. However, for the sake of concreteness, we show in figure Figure 3 what would result programming the above specified behaviours by using ReSpecT, firstly introduced in [13], as the reaction specification language. Since it is not part of the TuCSoN model definition, we do not discuss here in detail the ReSpecT language, forwarding the interested reader to [12].

4 Application Areas

TuCSoN is very well suited for all those Internet applications where the data to be accessed, as well as the computational activities involved, are characterised by a high degree of heterogeneity and dynamicity. In the rest of this Section, we shortly discuss three promising application areas for mobile agent applications, which should substantially benefit from TuCSoN-based coordination.

4.1 Distributed Information Retrieval

In the area of *distributed information retrieval*, mobile Internet agents can be exploited to move data elaboration capabilities through information sources, instead of transferring huge amounts of data. In this context, the TuCSoN model looks well-suited for the coordination of the interactions among mobile agents and local resources, since it effectively helps in dealing with the main problem of distributed information retrieval, that is, the heterogeneity of the information sources.

As in the library example presented above, the typical application environment of a mobile agent for information search and retrieval is a collection of Internet nodes characterised by different architectures, technologies, knowledge representation languages and systems, and managed in a highly decentralised way. This makes it impractical (if not impossible) to face heterogeneity by acting on the information sources. When direct communication models are adopted, this forces mobile agents to be aware of heterogeneity: they have to know how to behave in every different hosting environment where they have to execute. This makes agent design complex, and makes mobile agent application inflexible and practically closed to new information sources (which could imply a re-design of the agent).

The TuCSoN solution is then to mediate interaction, and to charge the burden of handling heterogeneity neither on the information sources nor on the agents, but on the coordination media they use to interact. This is obviously possible in general only when the behaviour of communication abstractions can be defined according to the expected agents' behaviours: in TuCSoN, any tuple centre can be locally programmed to take into account the peculiar structure of the local resources, and enabling mobile entities to access resources with a simple and straightforward protocol, independent of the current hosting environment. Somehow, the role that suitably programmed tuple centres assume in this context is similar the one of the "intelligent mediators" in the research area of heterogeneous databases integration [4].

For a detailed example of the use of TuCSoN in the the area of distributed information retrieval we forward the interested reader to [2].

4.2 Workflow Management

Workflow management [24] constitutes another interesting area for TuCSoN-coordinated mobile agents, where workflow rules can be seen as global coordination laws, to be naturally embedded over tuple centres.

In this context, it has already been shown how tuple spaces can be used as information repositories for the current state of the activities [34]. Tuple spaces spread over project sites contain tuples representing the tasks which can be performed according to the current project's state. An agent, specialised for a given kind of task, can explore these sites searching for available tasks, choose the most appropriate one on the basis of a given fitness function, and extract the corresponding tuple from the tuple space, in order to avoid duplicated task assignments. When the task is accomplished, tuples representing the subsequent tasks to be performed have to be put in tuple spaces, according to the new status of the project. The main drawback of this approach is that it requires agents to be aware of the workflow rules, in order to evaluate which new tasks can be performed after they have accomplished one. Since these are, *de facto*, coordination rules, and are typically quite dynamic in any workflow application, they should not be charged upon agents, but, more properly, on the coordination medium. However, tuple spaces do not allow any complex coordination policy to be implemented straightforwardly, as discussed in Section 2.

TuCSoN notably simplifies the management of the distributed activities from the agent side, by suitably defining tuple centre's behaviour. In fact, agents can interact with tuple centres by simply selecting a task, withdrawing the corresponding tuple, and, when the task is brought to end, notifying the tuple space with an appropriate tuple. Tuple centres, by their side, can be programmed so as to react by making available to the agents all and only those tuples representing the tasks which are available at any time of the computation, with respect to the current status of the project. In this way, workflow rules are totally embodied in tuple centres, and agents can disregard them and concentrate only on task choice and accomplishment.

4.3 Electronic Commerce

Agent-mediated electronic commerce [36] is another interesting area for TuCSoN applications, since tuple centres' computational capabilities can account for both the heterogeneity of information sources and the dynamics of commercial interaction. Let us consider an *buyer* agent that roam through specific commercial sites to look for a stock of given items, with the constraint of a maximal buying price. For each site, the *buyer* agent first checks for the availability of the needed item, then, in case it is found, the buyer starts a transaction with a local *seller* agent. When coordination is based on (non-programmable) tuple spaces, the local administrator must provide the tuples representing items to sell, along with their price and availability. A buyer agent reads tuples in the tuple space and evaluate the possible actions. Then, the buyer agent can insert in the tuple space a tuple representing its decision, such as a definitive or a conditioned decision to buy (implying, for example, a request of discount). The insertion of this tuple can awake a *seller* agent that evaluates the decision of the buyer and replies to it by inserting another tuple in the space. So on, until the two agents reach an agreement or a non-agreement. While the tuple space well suit the former product analysis phase, which is basically data-oriented, the latter contracting phase forces the contracting agents to communicate and exchange data via the tuple space in an indirect way, while direct communication would have better suited this phase.

Also in this case, the TuCSoN model can provide several advantages. In the former phase (analysis of the product), which is basically a problem of retrieval of dynamic and heterogeneous information, TuCSoN can be exploited as already discussed in Subsection 4.1. In the latter phase, a TuCSoN tuple centre can lead to more dynamic, efficient, and flexible interactions, by directly integrating contracting capabilities into the tuple centre in form of reactions. For example, the seller agent could specify the behaviour of a tuple centre providing for dynamic evaluation of the willing of the buyer agent as it attempts to read information from the tuple space. Thus, instead of giving a fixed price for its product, a tuple centre could dynamically set the product prices and the corresponding tuples, taking into account the buyer's requests. This would avoid complex peer-topeer contracting protocols between the buyer and the seller, simplifying the code of both. The buyer could evaluate the available information and take its decision, while the seller would simply perform the commercial transaction, and handle the delivering and payment procedures.

5 Conclusions

The most promising paradigm to design and program distributed applications over the Internet is based on network-aware and mobile agents. However, this new paradigm calls for new models and languages to manage interactions among the application agents.

The paper identifies the main characteristics of a coordination model suitable for Internet applications and presents the TuCSoN coordination model, based on independently programmable communication abstractions local to each node, called tuple centres. By defining their behaviour in response to communication events, tuple centres can be charged of many issues critical to Internet applications, such as heterogeneity and dynamicity of the execution environments, and (mobile) agent cooperation over space and time.

In spite of its many features, the TuCSoN model leaves some problems open. In fact, the real Internet structure is not a flat one, but can be considered a hierarchy of administrative domains separated by firewalls and gateways. Then, we feel that TuCSoN should be able to model this kind of topology, with regard both to the naming of the tuple centres and to their local programmability. Furthermore, security issues, such as agent authentication and privacy, should be directly supported by the model. Our research is currently focusing on these issues, which play a key role in the context of Internet applications.

References

[1] G. Agha and C.J. Callsen. Actorspace: an open distributed programming paradigm. *ACM Sigplan Notices*, 28(7):23–32, July 1993.

[2] The Authors. TuCSoN: a coordination model for mobile information agents. In D.G. Schwartz, M. Divitini, and T. Brasethvik, editors, *Proceedings of the First International Workshop on Innovative Internet Information Sytems (IIIS'98)*, pages 177–187, Pisa (Italy), June 8–9 1998. ISSN 0802-6394.

[3] J. Baumann, F. Hohl, N. Radouniklis, K. Rothermel, and M. Strasser. Communication concepts for mobile agent systems. In *Mobile Agents 97*, volume 1219 of *LNCS*, pages 123–135. Springer-Verlag, 1997.

[4] S. Bergamaschi and et. al. An intelligent approach to information integration. In *International Conference on Formal Ontology in Information Systems*, Trento, Italy, 1998.

[5] G. Cabri, L. Leonardi, and F. Zambonelli. The impact of the coordination model in the design of mobile agent applications. In *Proceedings of the 22nd Computer Software and Applications Conference*. IEEE CS Press, August 1998.

[6] G. Cabri, L. Leonardi, and F. Zambonelli. Reactive tuple spaces for mobile agent coordination. In *Proceedings of the 2nd Workshop on Mobile Agents*, volume 1477 of *LNCS*. Springer-Verlag, September 1998.

[7] L. Cardelli and A. D. Gordon. Mobile ambient, 1997. (To be found at `http://www.research.digital.com/SRC/personal/Luca_Cardelli/Ambit`).

[8] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, April 1989.

[9] P. Ciancarini. Distributed programming with logic tuple spaces. *New Generation Computing*, 12, 1994.

[10] P. Ciancarini, R. Tolksdorf, F. Vitali, D.Rossi, and A. Knoche. Coordinating multiagent applications on the WWW: A reference architecture. *IEEE Transactions on Software Engineering*, 24(5):362–375, 1998.

[11] E. Denti, A. Natali, and A. Omicini. Programmable coordination media. In *Coordination Languages and Models*, volume 1282 of *LNCS*, pages 274–288. Springer-Verlag, 1997.

[12] E. Denti, A. Natali, and A. Omicini. On the expressive power of a language for programming coordination media. In *Proceedings of the 1998 ACM Symposium on Applied Computing (SAC '98)*, Marriott Marquis, Atlanta, Georgia, U.S.A., February 27 - March 1 1998.

[13] E. Denti and A. Omicini. Designing multi-agent systems around an extensible communication abstraction. In A. Cesta and P.-Y. Schobbens, editors, *Proceedings of the 4th ModelAge Workshop on Formal Models of Agents, Certosa di Pontignano, Italy, January 15–18, 1997*, pages 87–97. National Research Council of Italy, 1997. To be published by Springer-Verlag in the LNAI Series.

[14] P. Domel, A. Lingnau, and O. Drobnik. Mobile agent interaction in heterogeneous environment. In *Mobile Agents 97*, volume 1219 of *LNCS*, pages 136–148. Springer-Verlag, 1997.

[15] R. Englemore and T. Morgan, editors. *Blackboard Systems.* Addison-Wesley, Reading, Mass., 1988.

[16] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *Proc. of the Third International Conference on Information and Knowledge Management*, Gaithersburg, Maryland, November 1994.

[17] A. Fuggetta, G. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5), May 1998.

[18] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1), January 1985.

[19] D. Gelernter. Multiple tuple spaces in Linda. In *Proceedings of PARLE*, volume 365 of *LNCS*, 1989.

[20] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, February 1992.

[21] M.R. Genesereth and R.E. Filkes. Knowledge interchange format: Version 3.0 reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992.

[22] R. Gray. Agent Tcl: A flexible and secure mobile-agent system. In *Proc. of the Fourth Annual Tcl/Tk Workshop*, Monterey, California, July 1996.

[23] IBM. T spaces, 1998. `http://www.almaden.ibm.com/TSpaces`.

[24] G. Kappel, S. Rausch-Schott, and W. Retschitzegger. Coordination in workflow management systems - a rule-based approach. In W. Conen and F. Neumann, editors, *Coordination Technology for Collaboration Application*, volume 1316 of *LNCS*. Springer-Verlag, 1998.

[25] N.M. Karnik and A.R. Tripathi. Design issues in mobile-agent programming systems. *IEEE Concurrency*, 6(3):52–61, 1998.

[26] J. Kiniry and D. Zimmerman. A hands-on look at Java mobile agents. *IEEE Internet Computing*, 1(4):21–33, July–August 1997.

[27] Sun Microsystems. The javaspace specifications, 1998. `http://www.javasoft.com/DistributedComputing`.

[28] N. Minsky and J. Leichter. Law-governed Linda as a coordination model. In *Object-Based Models and Languages*, volume 924 of *LNCS*, pages 125–145. Springer-Verlag, 1994.

[29] A. Ohsuga, Y. Nagai, Y. Irie, M. Hattori, and S. Honiden. PLANGENT: an approach to making mobile agents intelligents. *IEEE Internet Computing*, 1(3):50–57, 1997.

[30] OMG. CORBA 2.1 specifications, 1997. (`http://www.omg.org`).

[31] A. Omicini, E. Denti, and A. Natali. Agent coordination and control through logic theories. In *Topics in Artificial Intelligence*, volume 992 of *LNAI*, pages 439–450. Springer-Verlag, 1995.

[32] G.A. Papadopoulos and F. Arbab. Coordination models and languages. *Advances in Computers*, 46:The Engineering of Large Systems, August 1998. To appear.

[33] H. Peine and T. Stolpmann. The architecture of the Ara platform for mobile agents. In *Mobile Agents '97*, volume 1219 of *LNCS*, pages 50–61. Springer-Verlag, 1997.

[34] G. Piccinelli, F. Marcello, and G. Zugliani. An internet-based framework for federated process support. In D.G. Schwartz, M. Divitini, and T. Brasethvik, editors, *Proceedings of the First International Workshop on Innovative Internet Information Sytems (IIIS'98)*, pages 49–61, Pisa (Italy), June 8–9 1998. ISSN 0802-6394.

[35] A.I.T. Rowstron. *Bulk Primitives in Linda Run-Time Systems.* PhD thesis, The University of York, 1996.

[36] H. Vogler, M.L. Moschgath, and T. Kunkelmann. Enhancing mobile agents with electronic commerce capabilities. In M. Klusch and G. Weiss, editors, *Cooperative Information Agents II*, volume 1435 of *LNCS*. Springer-Verlag, 1998.

[37] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A note on distributed computing. In *Mobile Object Systems*, volume 1222 of *LNCS*, pages 49–64. Springer-Verlag, 1997.