TuCSoN: a Coordination Model for Mobile Information Agents

Andrea Omicini
LIA - DEIS - Università di Bologna
Viale Risorgimento, 2
Bologna, Italy 40136
aomicini@deis.unibo.it

Franco Zambonelli
DSI - Università di Modena
Via Campi 213b
Modena, Italy 41100
franco.zambonelli@unimo.it

Abstract

The increasing need to access and elaborate dynamic and heterogeneous information sources distributed over Internet calls for new models and paradigms for application design and development. The mobile agent paradigm promotes the design of applications where agents roam through Internet sites to locally access ad elaborate information and resources, possibly cooperating with each other. This paper focuses on mobile agent coordination, and discusses the TuCSoN coordination model for Internet applications based on mobile information agents. The model is based on the notion of *tuple centre*, a tuple-based interaction space associated to each site and to be used both for inter-agent cooperation and for accessing to local infromation sources. TuCSoN tuple centres enhance tuple spaces because their behaviour in response to communication events can be programmed. This can be used to deal with heterogeneity and dynamicity of the information sources, as well as to ensure some degree of global data integrity. The effectiveness of the TuCSoN model is shown by means of an application example in the area of Internet information retrieval.

1 Introduction

The increasingly growing Internet infrastructure as well as the almost pervasive diffusion of the WWW technology are changing the way Internet is conceived and exploited. Far from considering it as a raw communication media, the current challenge is to exploit Internet as a *globally distributed information system* where a wide range of distributed data and resources can be accessed and elaborated.

The above change of perspective outlines the inadequacy of traditional programming paradigms to meet this new scenario. Distributed applications composed of entities statically assigned to given execution environments and cooperating in a (mostly) network-unaware fashion cannot deal with the intrinsic dynamicity, unreliability and heterogeneity of Internet and of its information sources. The mobile agent paradigm suits the new application scenario: applications are composed by network-aware entities capable of dynamically changing their execution environment. The above change of perspective leads to several advantages in terms of saved bandwidth, dynamicity, and reliability [WWWK97]. For instance, in the area of distributed information retrieval, mobile agents may move locally to the data of interest to access and elaborate them, needing neither continuous network connection nor the transfer of large amounts of data.

While several new mobile agent systems and programming environments keep on appearing [KZ97], many issues still needs to be analysed before the mobile agent paradigm will gain wide acceptance [CGPV97]. A crucial point in this framework is represented by the model for agent interaction. Understanding how interactions can be ruled and exploited is the basic goal of current research on *coordination* languages and models [MC94, GC92]. The research interest in this area naturally derives from recognising the space of interaction as a source of complexity and expressive power for multi-component applications [Weg97]. From a software engineering viewpoint, the choice of a coordination model has an obvious influence over the design of complex applications, particularly when mobile agents are involved [CLZ98a]. Even more, we argue that a coordination model should effectively support application design and development, so that the component interaction space can fruitfully be exploited as an independent design space.

Following the classification introduced by [PA98], coordination models can be distinguished in two classes: *control-driven* and *data-driven* ones. The former class includes those models (such as ConCoord [Hol96], MANIFOLD [AHS93], and RAPIDE [LKA+95]) which focus on communication actions, mainly by ruling the topology of interaction between the applications components. Models of the latter class focus instead on communication information, by making coordination rules depend on information exchanged in the interaction. The most typical examples are blackboard-based models [EM88] and Linda [Gel85] with all of its descendents.

We argue that data-driven models better fit our application scenario than control driven ones. On the one hand, control-driven models do not cope well with highly autonomous, dynamic, and unpredictable coordinated entities like mobile agents. For instance, Internet applications based on mobile agents can hardly guarantee precise temporal and spatial schedules of communication actions. On the other hand, the intrinsic orientation of data-driven models towards communication information provides a more natural support to the coordination of information-oriented applications. In particular, a coordination model based on a shared dataspace like a Linda tuple space provides many key-features for the design and development of information-based systems, like associative access to information, as well as for mobile agent coordination, like temporal and spatial agent uncoupling.

Still, data-driven models typically lack control and flexibility required by today complex distributed applications. In particular, if interactions are tied to the pre-defined mechanisms of Linda, the burden of managing both information and component heterogeneity has to be charged over either mobile agents or local execution environments. This makes applications more difficult to be designed and incrementally modified, and increases their development costs.

The above considerations have led to several extensions of the raw Linda model towards extensibility and programmability, like $\mathcal{ACLT}$ [ODN95] and Law-Governed Linda [ML94]. The TuCSoN model (*Tu*ple *C*entres *S*pread *o*ver *N*etworks), described in Section 2, applies the same guidelines to the coordination of mobile agents. In TuCSoN, each local execution environment is built around a local communication abstraction, called *tuple centre*, which can be accessed by mobile agents through Linda-like communication primitives. However, unlike Linda tuple spaces, TuCSoN tuple centres are *programmable*, in that their behaviour in response communication events can be defined by means of an *ad hoc* specification language allowing *reactions* to be associated to communication operations. Reactions can be exploited to embody coordination laws into the coordination medium, and to make it possible to adaptively deal with heterogeneity, transparently to both mobile agents and components of the local execution environments. A sample *distributed information retrieval* application is then discussed in Section 3 to show the effectiveness of the TuCSoN approach in the context of Internet-based information systems.

2 The TuCSoN Coordination Model

The TuCSoN coordination model is aimed at supporting the design and development of information-oriented applications based on mobile agents. Its approach is then basically data-oriented, and exploits a peculiar notion of shared data space *à la* Linda.

In order to cope with mobility, a node of the network that wants to accept mobile agents must define its own *local* communication space, used by agents to interact with other agents as well as with the local execution framework. Each local communication space consists of a multiplicity of *global* (w.r.t. the node) communication abstractions, called *tuple centres*.

Roughly speaking, a tuple centre is an enhanced tuple space, denoted by a name which is locally (i.e., w.r.t. a node) unique. Each tuple centre can be accessed by mobile agents via a Linda-like communication interface. The communication primitives `out`, `in` and `rd`, as well as their non-blocking versions `inp` and `rdp`, have basically the same semantics of their Linda counterparts. Each operation has to be performed with respect to one specific tuple space, by explicitly denotating it by name. If `t` is a tuple centre identifier, for instance, an `out(`*`T`*`)@t` operation performed by an agent express its intention to insert tuple *`T`* into the tuple centre denoted by `t` in the local communication space of the node where it is currently executing.

This allows a very expressive form of communication, by providing a multiplicity of communication channels which can be accessed and programmed individually and independently. However, this feature is not strictly relevant to our purposes here, so we will leave it aside for the rest of the paper, thus not discussing the advantages of having many tuple centres in one node. In the following, any communication operation and reaction specification will then be implicitly referred to the *default tuple centre*.

2.1 Tuple Centres

A typical problem of Linda-like coordinated systems relies on the tuple space 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 the application purposes, or the interacting entities have to be charged of the burden of the coordination. That is why the TuCSoN approach goes over Linda, by providing for flexibility and control via a *programmable coordination abstraction* [DNO97] called tuple centre. Unlike Linda tuple spaces, the behaviour of a tuple centre as a communication channel can be defined and tailored to the application needs.

The behaviour of a (stateful) communication abstraction like a shared data space is easily defined as the observable state transition following a communication event. As a result, defining a new behaviour for a tuple centre means essentially specifying a new state transition in response to a standard communication event. Whichever model is adopted for this enhancement, it has to accomplish some fundamental requirements. First, it should allow any communication event to be intercepted and make all its related information available, thus raising observability from communication state (as in Linda tuple spaces) to communication operations. Then, it should make it possible to access and manipulate the state of the communication abstraction (e.g. the content of a tuple centre in terms of tuples).

2.2 TuCSoN Reactions

In order to accomplish the above requirements, any of the TuCSoN basic communication primitives (`out`, `in`, `rd`, `inp`, `rdp`) can be associated with a specific computational activity, called *reaction*. A reaction is defined as a set of non-suspensive operations, with a success/failure transactional semantics: successful reactions may atomically produce effects on the tuple centre state, failed reactions yield no result at all. Moreover, 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, before any other agent-triggered communication event is served. As a consequence, from the agents' viewpoint, the result of the invocation of a communication primitive is the sum of the effects of the primitive itself and of all the reactions it triggered, perceived altogether as a single-step transition of the tuple centre state. Such 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.

Programming reactions asks for the definition of a suitable specification language meeting the fundamental requirements sketched in Subsection 2.1. The language should provide access to all the information related to communication events (such as the performing agent, the operation performed, the tuple required, ...), and make it possible for reactions to freely access and modify tuple space information. In addition, we argue that only a Turing-powerful language would in principle ensure that the computational load can be distributed by the system designer between the coordinated entities and the coordination media according to abstract design criteria only.

The TuCSoN abstract model for the coordination of mobile agents prescinds from the language adopted for reaction specification, and also from both the kind of tuple used for the communication and the tuple matching criterion. In this paper we adopt first-order logic tuples, along with unification as the corresponding matching mechanism, and take ReSpecT (introduced in [DNO98]) as the specification language of choice. The former choice brings into the model the expressiveness of first-order logic (which arguably well copes with information systems), and makes it possible the twofold interpretation of the tuple space as both a message repository and a theory of the communication. The latter choice satisfies all the language requirements introduced above, and allows the TuCSoN model to be immediately tested in practice, since ReSpecT is already exploited by the well-established $\mathcal{ACLT}$ system for distributed multi-agent applications [ODN95].

2.3 Programming Reactions with ReSpecT

A ReSpecT tuple like `reaction(`*`Op`*`,`*`Body`*`)` associates a communication operation *`Op`* to the *`Body`* reaction body. Whenever an operation *`Op`* is performed, all its corresponding reactions are triggered, and their reaction bodies are executed. Each reaction body is syntactically defined as a conjunction of *reaction goals*. which can access the information related to the triggering communication event (through predicates like `current_agent/1`, `current_op/1`, ...), manipulate terms (through term predicates like term equality/inequality, term unifiability/non-unifiability, ...), as well as access and modify the contents of the tuple centre in terms of tuples (through predicates like `out_r`, `in_r`, `rd_r`, `no_r`, ...). In particular, `out_r` basically works as a conventional `out`, while `in_r` and `rd_r` have the same effects as `inp` and `rdp`, respectively. Instead, `no_r` succeeds in case its argument's tuple does not unify with any tuple in the tuple space, fails otherwise.

Reactions can be defined not only for communication primitives, but also for operations over tuple spaces performed inside reactions. As a result, in a tuple `reaction(`*`Op`*`,`*`Body`*`)`, *`Op`* may be not only `out(`*`T`*`)`, `in(`*`T`*`)`, `rd(`*`T`*`)`, `inp(`*`T`*`)`, or `rdp(`*`T`*`)`, but also `out_r(`*`T`*`)`, `in_r(`*`T`*`)`, `rd_r(`*`T`*`)`, or `no_r(`*`T`*`)`, where *`T`* is a meta-variable standing for a logic tuple.

Further reaction predicates are related to the peculiar semantics of some communication primitives. First of all, unlike the `out` primitive, `in` and `rd` may be seen (see also [GZ97]) as made of two distinct communication events: the first query phase (called the *pre* phase), when a tuple template is provided, and the subsequent answer phase (called the *post* phase), when a unifying tuple is eventually returned to the querying agent. According to that, ReSpecT introduces the two `pre/0` and `post/0` predicates, which obviously succeed only in the *pre* and *post* phase, respectively. As a result, we may define reactions which will be actually executed in the *pre* or *post* phase only of an `in` or `rd` operation. Analogous considerations apply to `inp` and `rdp`, too. Since, however, these primitives may either succeed or fail, ReSpecT introduces two further predicates, `success/0` and `failure/0`, which, as intuitive, succeed only in the case that the current non-blocking primitive succeeded or failed, respectively. This makes it possible to define different reactions for the *post* phase of an `inp` or a `rdp`, depending on the success or failure of the operation. We forward the interested reader to [DNO98] for a wider presentation of ReSpecT.

3 An Application Example

As our application example, we consider an information system consisting of a collection of WWW servers spread over a WAN, whose structure is heterogeneous both in terms of SW and HW architectures and of information kind and organisation. For instance, think of a group of servers spread over a tourist area, containing information about museums, hotels, sport centres, tourist attractions, and so on. In this framework, we consider a mobile agent application in charge of gathering information from HTML pages and automatically producing as its output a page of references on its home site. Mobile agents are used as 'knowledge retrievers', spread over the WAN with the aim of roaming across the servers, gathering all relevant information, and coming back to their home site.

3.1 Heterogeneity

The issue of the heterogeneity of information sources may be faced in principle by making mobile agents aware of all the different ways in which knowledge is represented. This choice, however, would make agent design a highly complex task, and produce a very inflexible structure, where any new server to be added may imply a re-design of the agents.

In TuCSoN, the capability of programming the communication abstraction allows the burden of heterogeneity to be charged upon tuple centres, bridging the gap between agent interaction protocols and the peculiar model for knowledge representation adopted by each site. Mobile agents can then be designed independently of any specific choice for information organisation, and according to a simple and straightforward interaction protocol. For instance, in our application example, each agent is initially given a set of keywords, with the goal of retrieving the URL of every HTML page concerning each keyword. Then, agents simply ask for tuples of the form `kwURLs(`*`KW`*`,`*`URLs`*`)` through a `rd` operation, independently of the hosting execution environment. Keyword *`KW`* is provided as a constant by the agent, which expects as a result the instantiation of variable *`URLs`* to the list of all URLs of the HTML pages containing some references to *`KW`*.

This simple agent interaction protocol is intended to work on every server, independently of the server model for knowledge representation. Take for instance two WWW sites, A and B, recording page-content relations in two different ways. Server A describes the content of each page in term of keywords by means of tuples of the form `keyword(`*`KW`*`,`*`PageName`*`)`, while page organisation is recorded by means of tuples of the form `page(`*`PageName`*`,`*`PageURL`*`)`.

```
% server A
reaction(rd(kwURLs(KW,_)), ( pre, no_r(kwURLs(KW,_)), out_r(kwPages(KW,[])) )).
reaction(out_r(kwPages(KW,Pages)), in_r(kwPages(KW,Pages))).
reaction(out_r(kwPages(KW,Pages)), ( in_r(keyword(KW,Page)),
    out_r(kwPages(KW,[Page|Pages])) )).
reaction(out_r(kwPages(KW,Pages)), ( no_r(keyword(KW,_)),
    out_r(pageURLs(KW,Pages,[])) )).
reaction(out_r(pageURLs(KW,Pages,URLs)), in_r(pageURLs(KW,Pages,URLs))).
reaction(out_r(pageURLs(KW,[Page|Pages],URLs)), ( out_r(keyword(KW,Page)),
    rd_r(page(Page,URL)), out_r(pageURLs(KW,Pages,[URL|URLs])) )).
reaction(out_r(pageURLs(KW,[],URLs)), out_r(kwURLs(KW,URLs))).
% server B
reaction(rd(kwURLs(KW,_)), ( pre, no_r(kwURLs(KW,_)), out_r(query(kwSearch,KW)) )).
reaction(out(answer(kwSearch(KW),URLs)), (
    in_r(answer(kwSearch(KW),URLs)), out_r(kwURLs(KW,URLs)) )).
```

Figure 3.1

In order to cope with agent protocol, the tuple centre of server A will be programmed to react to a `rd(kwURLs(`*`KW`*`,`*`URLs`*`))` operation *(i)* by checking whether the required tuple is present, *(ii)* in case it is not, by finding all the pages referring to keyword *`KW`*, *(iii)* by building the list of their corresponding URLs, and *(iv)* by finally providing the agent with the required answer tuple.

Instead, server B is built around a DBMS application recording the site content. The DBMS interacts with the tuple centre 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, a query of the form `query(kwSearch(`*`KW`*`))` makes the wrapper ask the DBMS to return the URLs of all the pages containing references to keyword *`KW`*. The consequent answer is then given as a tuple `answer(kwSearch(`*`KW`*`),`*`URLs`*`)`. Correspondingly, once detected the absence of the tuple required, the tuple centre of the server B reacts to a `rd(kwURLs(`*`KW`*`,`*`URLs`*`))` operation by producing a `query(kwSearch(`*`KW`*`))` tuple for the wrapper. When the corresponding tuple `answer(kwSearch(`*`KW`*`),`*`URLs`*`)` is inserted by the wrapper, the `kwURLs(`*`KW`*`,`*`URLs`*`)` tuple initially required by the agent is finally produced. The ReSpecT code for servers A and B is shown in Figure 3.1. Despite the different choices for the representation of page-content relations, mobile agents interact with both servers A and B in the same way, by simply asking for tuples `kwURLs(`*`KW`*`,`*`URLs`*`)`. In fact, both tuple centres actually provide agents with the same uniform view of such information: their behaviour make them be perceived by any agent as they would contain a tuple `kwURLs(`*`KW`*`,`*`URLs`*`)` for any possible keyword *`KW`*.

3.2 Integrity

Malicious or simply badly programmed agents may undermine the internal coherence of the hosting execution environment. A knowledge source should then be protected from those interactions possibly affecting the semantic consistence of the information contained. In a coordination model like TuCSoN where communication is mediated by a shared dataspace, the global consistency of a system w.r.t. interaction can be granted by providing for the consistency of the communication state (the tuple centre state, in TuCSoN).

Take again the server A sketched in Subsection 3.1, where page organisation is recorded by tuples of the form `page(`*`PageName`*`,`*`PageURL`*`)`. As an example, we may like to ensure that, at any time, each HTML page of the server has its own unique URL recorded in the tuple centre. This means that, given a *`PageName`* page, it should never happen that no

```
% server A
reaction(out(updatePage(Name,NewURL)), ( in_r(updatePage(Name,NewURL)),
    in_r(page(Name,_)), out_r(page(Name,NewURL)) )).
reaction(out(page(PageName,URL)), in_r(page(PageName,URL))).
reaction(in(page(_,_)), ( post, current_tuple(page(PageName,URL)),
    out_r (page(PageName,URL)) )).
reaction(inp(page(_,_)), ( post, success, current_tuple(page(PageName,URL)),
    out_r (page(PageName,URL)) )).
```

Figure 3.2

page(*PageName*,*PageURL*) tuple occurs in the tuple centre, and, dually, that more than one tuple of that kind occur at the same time.

The behaviour of the tuple centre should then be defined so as to avoid insertions and removals of page/2 tuples. In addition, when a page location is to be updated, it should allow such tuples to be replaced with new ones atomically, so as to preserve information consistency at any time. Correspondingly, the ReSpecT code shown in Figure 3.2 makes any in, inp, or out of page/2 tuples ineffective. Moreover, it makes it possible to atomically update the URL of a page *PageName* to *NewURL* by means of the emission of an updatePage(*PageName*,*NewURL*) tuple.

3.3 Dynamicity

Today information systems are typically characterised by a high degree of dynamicity. For any single information source, new information can be gathered from the outside, new knowledge can be inferred from the old one, new components can be added to provide new services and capabilities. The burden of managing dynamicity may be charged over mobile agents, but this, again, would make them unnecessarily complex. Instead, in this case too, a more effective solution is to program the tuple centre so as to deal with the dynamicity of the information sources.

Let a server C have the same structure and organisation of server A described in Subsections 3.1–3.2. Then, suppose that an intelligent agent $\mathcal{L}$ is added to C, to learn from user interaction and infer new information from user's exploration paths. In particular, say that $\mathcal{L}$ is able to infer that two knowledge items are strictly related from a semantic viewpoint. So, whenever such a new relation is inferred, $\mathcal{L}$ makes it available to the world by emitting a tuple of the form relKW(*KW*,*RKW*), stating that whichever refers to *RKW* typically refers to *KW* too. It would then be desirable to be able to integrate the new knowledge inferred by $\mathcal{L}$ in the whole system and make it usable by mobile agents in a transparent way, without affecting agents' interaction protocol. To this end, a TuCSoN tuple centre may be exploited so that, when an agent moves to C and asks for keyword *KW*, it transparently gets all the pages concerning both *KW* and all its related keywords, as inferred by $\mathcal{L}$ so far.

Even though quite sophisticated policies could be built in the system, we will give here a very simple example. Whenever a new relKW(*KW*,*RKW*) tuple is inserted by $\mathcal{L}$, the tuple centre reacts by looking for all the pages referring to *RKW*. For every such a *PageName* page found, a tuple keyword(*KW*,*PageName*) is added, which states that *PageName* concerns *KW*, too. Finally, any information previously produced about *KW* in form of a kwURLs(*KW*,*URLs*) tuple is removed, so that a subsequent request by any agent will cause the production of the updated information according to the code shown in Subsection 3.1. The corresponding ReSpecT code is shown in Figure 3.3.

```
% server C
reaction(out(relKW(KW,RKW)), out_r(newPages(KW,RKW,[]))).
reaction(out_r(newPages(KW,RKW,Pages)), in_r(newPages(KW,RKW,Pages))).
reaction(out_r(newPages(KW,RKW,Pages)), ( in_r(keyword(RKW,Page)),
    out_r(newPages(KW,RKW,[Page|Pages])) )).
reaction(out_r(newPages(KW,RKW,Pages)), ( no_r(keyword(RKW,_)),
    out_r(relPages(KW,RKW,Pages)) )).
reaction(out_r(relPages(KW,RKW,Pages)), in_r(relPages(KW,RKW,Pages))).
reaction(out_r(relPages(KW,RKW,[Page|Pages])), (
    out_r(keyword(KW,Page)), out_r(keyword(RKW,Page)) )).
reaction(out_r(relPages(KW,RKW,[])), in_r(kwURLs(KW,_))).
```

Figure 3.3

4 Related Works

In the last few years, several systems and programming environments have been proposed to support the development of Internet applications based on mobile agents [RPZ97, VT97]. Despite this fermenting activity, only a few proposals focus on coordination issues.

Most of the proposed systems rely on message passing for inter-agent coordination and on the client-server model for the accesses to the local information systems. Typical examples are Java-based mobile agents systems, where communication occurs via the usual object-oriented interaction mechanisms, and low-level message-passing can be also exploited via TCP/IP [KZ97]. However, as argued in [CLZ98a], these models hardly suit Internet applications based on mobile agents because they do not promote interaction locality, and enforce a strict coupling between the interacting entities in terms of required mutual knowledge and temporal synchronisation.

In the meeting-oriented model, implemented by the Ara system [PS97], interaction occurs in the context of "meeting places". An active entity must assume the role of meeting initiator to open and define a meeting point. Agents join known meeting points and can there communicate and synchronise with the other agents participating in it. As a benefit of this model, interaction locality can be enforced by locally constraining meetings: a meeting takes place at a given execution environment and only local agents are allowed to participate. However, this model of coordination has some drawbacks. For instance, it requires both temporal synchronization between the interacting entities, and mutual knwoledge of meeting points. Furthermore, interaction with local resources is still basically tied to a client-server model (always-opened meetings abstract the role of servers), so that access to information sources is essentially ruled in a control-driven manner.

The concept of anonymous interaction via shared dataspaces associated to each local execution environment enforces locality in interactions as well as temporal uncoupling. A blackboard-based model of interaction is exploited both by the Ambit formal model for mobile computation [CG97] and by the ffMain system [DLD97] for mobile information agents. Still, the absence of any associative mechanism in accessing the data space, makes the model lack the degree of flexibility required for both inter-agent coordination and heterogeneous information source access. In fact, associative mechanisms allow the dataspace to be accessed according to information-driven policies, that is, based on information content, rather than on data structure and representation, or on some sort of labelling (such as memory addresses, or keys, and so on). This need for associative mechanisms in the context of mobile agent applications has been addressed by the Jada system [CR97]. Mobile agents can access to local tuple space to retrieve, in an associative way (i.e., by class name) object references.

The notions of programmable coordination medium [DNO97], in general, and of programmable tuple space, in particular, have already been proposed and applied in different contexts. For instance, $\mathcal{ACLT}$ [ODN95] exploits extensible tuple spaces [DNOV96a] for the coordination of distributed applications based on intelligent heterogeneous agents [DNOV96b]. Law-governed Linda [ML94] extends the tuple space model towards security and efficiency of the communication in multi-component distributed systems. However, in the context of mobile agents, MARS (Mobile Agent Reactive Spaces) [CLZ98b] is the only research proposal we have knowledge of which exploits programmable tuple spaces for mobile agent coordination. Actually developed in the context of an affiliated reasearch project, MARS significantly differs from TuCSoN in terms of both its tuple space model and its reaction model. In particular, MARS is mostly oriented to network management duties, rather than to help access to highly dynamic and heterogeneous information sources.

5 Conclusions

The mobile agent paradigm can be exploited to support the design and the development of efficient applications for the access to heterogeneous and dynamic information sources distributed over Internet. However, since the interaction space represents the core of the design process for information-oriented mobile agent applications, only the choice of an appropriate coordination model may allow a full exploitation of the peculiar advantages of the paradigm.

This paper presents TuCSoN, a coordination model for Internet applications based on mobile agents, and shows its effectiveness in the area of distributed information retrieval. TuCSoN exploits the notion of tuple centre as a programmable coordination medium whose behaviour can be defined and used to deal with heterogeneity and dynamicity of the information sources, as well as to ensure some degree of global data integrity. This simplifies the design of information-oriented applications, since most of the global system properties can be charged over a global communication abstraction without affecting mobile agents' design.

Future work will address several issues neglected by this paper:

- how and to which extent can TuCSoN reaction be allowed to access and modify the state of the associated tuple centre? What security issues such capabilities would involve?

- how and to which extent can mobile agents be given the capability of programming reactions to specific sites? And, again, what security issues such capabilities would involve?

Investigations in different application areas, such as electronic commerce and computer support cooperative work, would provide additional important feedbacks to complete and improve both the model and its implementation.

References

[AHS93] F. Arbab, I. Herman, and P. Spilling. An overview of Manifold and its implementation. *Concurrency: Practice and Experience*, 5(1):23–70, 1993.

[CG97] L. Cardelli and A.D. Gordon. Mobile Ambient, 1997. `http://www.research.digital.com/SRC/personal/Luca_Cardelli/Ambit`.

[CGPV97] G. Cugola, C. Ghezzi, G. Picco, and G. Vigna. Analyzing mobile code languages. In *Mobile Object Systems*, volume 1222 of *LNCS*, pages 94–109. Springer-Verlag, 1997.

[CLZ98a] G. Cabri, L. Leonardi, and F. Zambonelli. The impact of the coordination model in the design of mobile agent applications. In *Proceedings of the Computer Software and Applications Conference*. IEEE CS Press, August 1998.

[CLZ98b] G. Cabri, L. Leonardi, and F. Zambonelli. Reactive tuple spaces for mobile agent coordination. Technical Report DSI-03-98, University of Modena, February 1998. Submitted for Publication.

[CR97] P. Ciancarini and D. Rossi. Jada - coordination and communication for Java agents. In *Mobile Object Systems*, volume 1222 of *LNCS*, pages 213–226. Springer-Verlag, 1997.

[DLD97] 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.

[DNO97] 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.

[DNO98] 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.

[DNOV96a] E. Denti, A. Natali, A. Omicini, and M. Venuti. An extensible framework for the development of coordinated applications. In *Coordination Languages and Models*, volume 1061 of *LNCS*, pages 305–320. Springer-Verlag, 1996.

[DNOV96b] E. Denti, A. Natali, A. Omicini, and M. Venuti. Logic tuple spaces for the coordination of heterogeneous agents. In *Frontiers of Combining Systems*, pages 147–160. Kluwer Academic Publishers, 1996.

[EM88] R. Englemore and T. Morgan, editors. *Blackboard Systems*. Addison-Wesley, Reading, Mass., 1988.

[GC92] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, February 1992.

[Gel85] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1), January 1985.

[GZ97] D. Gelernter and L. Zuck. On what Linda is: Formal description of Linda as a reactive system. In *Coordination Languages and Models*, volume 1282 of *LNCS*, pages 187–204. Springer-Verlag, 1997.

[Hol96] A.A.. Holzbacher. A software environment for concurrent coordinated programming. In P. Ciancarini and C. Hankin, editors, *Coordination Languages and Models*, volume 1061 of *LNCS*, pages 249–266. Springer-Verlag, 1996. First International Conference, COORDINATION'96, Cesena, Italy, April 15–17, 1996.

[KZ97] J. Kiniry and D. Zimmerman. A hands-on look at Java mobile agents. *IEEE Internet Computing*, 1(4):21–33, July–August 1997.

[LKA$^+$95] D.C. Luckham, J.J. Kenney, L.M. Augustin, J. Vera, D. Brian, and W. Mann. Specification and analysis if system architecture using Rapide. *IEEE Transactions on Software Engineering*, 21(4):336–355, 1995.

[MC94] T.W. Malone and K. Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26:87–119, 1994.

[ML94] 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.

[ODN95] 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.

[PA98] G.A. Papadopoulos and F. Arbab. Coordination models and languages. *Advances in Computers*, 46, 1998. To appear.

[PS97] 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.

[RPZ97] K. Rothermel and R. Popescu-Zeletin, editors. *Proceedings of the First International Workshop on Mobile Agents*, number 1219 in LNCS. Springer-Verlag, April 1997.

[VT97] J. Vitek and C. Tschudin, editors. *Mobile Object Systems*, number 1222 in LNCS. Springer-Verlag, April 1997.

[Weg97] P. Wegner. Why interaction is more powerful than computing. *Communications of the ACM*, 40(5):80–91, May 1997.

[WWWK97] 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.