

Coordination in Open and Dynamic Environments via TuCSoN Semantic Tuple Centres

Elena Nardini

ALMA MATER STUDIORUM – Università di Bologna
`elena.nardini@unibo.it`

December 4, 2009



- 1 Motivations
- 2 Semantic Tuple Centres
- 3 TuCSoN
- 4 Semantic Tuple Centres in TuCSoN
- 5 Example of ReSpecT Tuple Centre Usage
- 6 Open points
- 7 The MediCoordination Application
- 8 Distributed TBox and ABox
- 9 MSL Service Coordination via Semantic Tuple Centres
- 10 Bibliography



Outline

- 1 Motivations
- 2 Semantic Tuple Centres
- 3 TuCSoN
- 4 Semantic Tuple Centres in TuCSoN
- 5 Example of ReSpecT Tuple Centre Usage
- 6 Open points
- 7 The MediCoordination Application
- 8 Distributed TBox and ABox
- 9 MSL Service Coordination via Semantic Tuple Centres
- 10 Bibliography



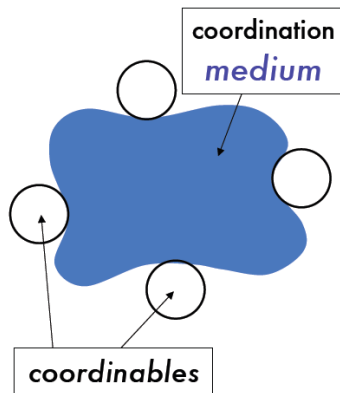
Modern Software Systems' Features

- Modern software systems like pervasive systems, internet applications, and web services, are mainly characterised by three features:
 - ▶ Distribution
 - ▶ Openness
- Such features call for coordination models able to provide **uncoupling** among system components.
- This is the underlying motivation that leads current research trends in the area of coordination middleware to adopt **Linda-like tuple spaces** as basic coordination abstractions [Nixon et al., 2008].



Coordination Meta-model

- Coordination is the activity of **managing / constraining** the software components interaction space, through the adoption of first class abstractions, from design to runtime.
- These first class abstractions (**coordination media**), enable interaction (including communication) among software components (**the coordinables**), and govern it by means of suitable **coordination laws** [Ciancarini, 1996].



Linda Model I

- Coordination medium: **Tuple Space**.
 - ▶ Multiset / bag of data object / structures called **tuples**.
- Communication Language:
 - ▶ **Tuple**: ordered collection of (possibly heterogeneous) information items.
 - ▶ **Templates**: specification of set / classes of tuples.
 - ▶ **Tuple Matching Mechanism**: mechanism to match tuples and templates.
- Coordination Language: set of **operations out**, **in**, and **rd**, to respectively put, read, and consume associatively tuples to / from the space.



Linda Benefits I

- Linda it is based on **generative communication**: tuple generated by a tuple producer has an independent existence in the tuple centre [Rossi et al., 2001, Nixon et al., 2008].
- Generative communication leads to **communication orthogonality**: both senders and receivers can interact even without having prior knowledge about each others.
 - ⇒ **Space uncoupling** (distributed naming) among coordinables.
 - ⇒ **Time uncoupling** among coordinables.



Linda Benefits II

- In Linda the tuple space access is **associative**: tuples are accessed through content and not address [Rossi et al., 2001, Nixon et al., 2008].

⇒ **Data-reference uncoupling.**

⇒ Generative communication and associative access make the tuple space model interesting for the engineering of system coordination in distributed and open applications.



Towards Tuple Centres I

- The behaviour of Linda tuple space is set once and for all by the model, and cannot be tailored to the specific application needs [Omicini and Denti, 2001].
- ⇒ Any coordination policy not directly supported by the model has typically to be charged upon system coordinables growing their complexity, especially in distributed, open and dynamic scenarios [Omicini and Zambonelli, 1999].



Towards Tuple Centres II

- Tuple centres extend the basic Linda tuple space model by making the behavior of the space programmable so as to encapsulate coordination laws directly in the coordination media.
 - ⇒ Higher decoupling among the system coordinables.
 - Tuple centres are thought to mediate interaction both among system coordinables and between coordinables and system resources (e.g., printer), acting as kinds of virtual proxies (or interfaces) of the resources and embedding coordination policies governing their exploitation.
- ⇒ Tuple centres can be conceived general-purpose customisable coordination media, as persistent entities of the system, mediating interaction among all system components and providing coordination functionalities to support coordinable cooperative activities.



Towards Semantic Tuple Centres I

- The main mechanism enabling coordination through a tuple centre, is the **tuple matching mechanism**:
 - ▶ coordination laws embedded in a tuple centre are enacted through the tuple matching mechanism, against the invocation of a communication primitive for inserting, reading, or consuming information items by means of tuples.
- Although the tuple centre model does not assume any matching mechanism [Omicini, 1999, Omicini and Denti, 2001], like the Linda implementation case, tuple centre implementations adopt a purely syntactic matching mechanism [Omicini and Denti, 2001].



Towards Semantic Tuple Centres II

- As already observed for [service-oriented applications](#) in the context of Pervasive Computing [Bandara et al., 2008] and Web Services [Paolucci et al., 2002], syntactic matching mechanisms pose several limitations in dealing with open and dynamic scenarios where the information exchanged could have different syntactic structures.
 - In our context it may mean that a system component (or also a system resource) refers to the concept [Car](#) and in the shared spaces there is information about [SportCar](#) or [CityCar](#) concepts – that are both types of [Car](#) –, with the syntactic matching mechanism it is not possible to match those concepts.
- ⇒ It could become difficult or not possible to coordinate heterogeneous and dynamic system coordinables.



Towards Semantic Tuple Centres III

- The advent of the Semantic Web has increased the interest in the use of semantic description of information through an **ontology language** and the use of **logical reasoning** over such descriptions to support information matching [Bandara et al., 2008, Paolucci et al., 2002].
 - An ontology language coupled with a logical reasoning offers two main advantages:
 - ▶ the ability to identify a match between logically equivalent information with syntactically different structures;
 - ▶ the ability to dynamically acquire, extend, and adapt the vocabulary used to describe the knowledge.
- ⇒ Such benefits are necessary within open scenarios where no assumptions can be made about the available knowledge.



Towards Semantic Tuple Centres IV

- The introduction of a semantic matching mechanism in tuple centres becomes desirable.
- Through sharing domain ontologies it is possible to face:
 - ▶ openness problems concerning heterogeneous coordinables that refer to information that is semantically equivalent but has a different syntactic representation;
 - ▶ dynamism problems concerning the inability of coordinables to have a prior knowledge about all possible information stored in a tuple centre;
 - ▶ the dynamic evolution of the application domain, by dynamically adapting the domain ontology.



Outline

- 1 Motivations
- 2 Semantic Tuple Centres
- 3 TuCSoN
- 4 Semantic Tuple Centres in TuCSoN
- 5 Example of ReSpecT Tuple Centre Usage
- 6 Open points
- 7 The MediCoordination Application
- 8 Distributed TBox and ABox
- 9 MSL Service Coordination via Semantic Tuple Centres
- 10 Bibliography



Existing Approaches

- **Semantic tuple space computing** is aimed at augmenting tuple spaces with semantic techniques [Nixon et al., 2008].
- Introduced for coping openness and dynamism in the context of Semantic Web and Semantic Web Services by means of the use of the tuple space coordination model, since it promotes decoupling among system coordinables.
- Proposes extensions of classical Linda-based systems with semantically enriched tuple spaces, designed specifically for the Semantic Web and Semantic Web Services.
- The proposed solutions are not well suited to deal with scenarios different from Semantic Web and Semantic Web Services:
 - ▶ provide abstractions depending from the chosen semantic technology;
 - ▶ suffer of the tuple space model limits.



Our Aim

- We assume tuple centre as coordination model to manage system interactions,
- with the aim to semantically enrich such model in order to provide a general-purpose and customisable coordination media also able to cope requirements like openness and dynamism,
- maintaining the model identity and without any assumption about the application context.



Ontologies and Individuals in Tuple Centres I

- From an abstract viewpoint, a tuple centre can be seen as a knowledge repository structured in a set of tuples.
 - In a semantic view, such knowledge represents a set of objects occurring in the application domain, whose meaning is described by the domain concepts and the description of relations among such concepts, within an ontology.
- ⇒ The two first ingredients to design the semantic tuple centre model are:
- ▶ **domain ontology** allowing to interpreter the semantic associated to the knowledge (set of tuples) stored into a tuple centre;
 - ▶ **domain objects** – represented by tuples – described so that they can be interpreted in a semantic way, by means of the domain ontology.



Ontologies and Individuals in Tuple Centres II

- In order to formally define the domain ontology and objects in the context of a tuple centre, the literature makes available a family of knowledge representation formalisms called **Description Logic (DL)** [Baader et al., 2003].
- By adopting such formalism, then we can exploit a DL-based language, that is a logic – i.e., formal language with well defined semantics – allowing [Baader et al., 2003]:
 - ▶ ontologies, and information using vocabulary defined by ontologies, to be shared and exchanged without disputes as to precise meaning;
 - ▶ automated reasoning techniques over ontologies that can be exploited – for example by intelligent agents – to face the dynamic evolution of the vocabulary used to describe the knowledge.



Ontologies and Individuals in Tuple Centres III

- In particular, we exploit the SHOIN(D) Description Logic formalism in order to model a semantic tuple centre in terms of domain ontologies and objects, since it represents:
 - ▶ a good compromise between expressiveness and complexity [Horrocks et al., 2003];
 - ▶ the theoretical counterpart of **OWL DL**, that is one of the three species of OWL [Horrocks et al., 2003]—the W3C standard ontology description language for Semantic Web. As standard OWL well fits the openness requirement.



Domain Ontology Description I

- DL allows to describe the tuple centre ontology in the form of a **terminology** through a so called **TBox**.
- TBox consists of **concept descriptions**, which denote sets of objects called **individuals**, and **role** descriptions, which denote binary relationships between individuals.
- In order to express such descriptions, SHOIN(D) provides a set of constructors described in [Horrocks et al., 2003], where the main of them are:

SHOIN(D)	Meaning
$C \sqcap D$	Intersection
$C \sqcup D$	Union
$\neg C$	Negation
$\forall R.C$	Value Restriction
$\exists R.C$	Existential Quantification
$\leq N$	Number Restriction
$\geq N$	Number Restriction



Domain Ontology Description II

- Moreover, TBox formalism provides the axioms $\sqsubseteq$ (Inclusion) and $\equiv$ (Equality), in order to define a taxonomy among concepts or roles.
- To give an explicative example of the TBox formalism use, we can think to describe the knowledge stored in a tuple centre as a set of individuals belonging to the *car* domain.
- We can define the following assertions:

$$Maker \sqsubseteq \top$$

$$Car \sqsubseteq (= 1 \text{ hasMaker})$$

$$(= 1 \text{ hasMaker}) \sqsubseteq Car$$

$$\top \sqsubseteq \exists \text{hasMaker.Maker}$$

$$CityCar \sqsubseteq Car$$



Domain Individual Description I

- Having understood how to describe the domain ontology of a tuple centre, the next step is to understand how to describe the domain objects of a tuple centre in form of tuples, so to obtain **semantic tuples**.
- For this purpose, DL defines through a so called **ABox** the **assertions** about the individuals and roles, in terms of the terminology defined through the TBox.



Domain Individual Description II

- ABox defines two kinds of assertions:

$$C(a) \quad R(b,c)$$

- Taking the example about the **car** domain, the ABox formalism [Baader et al., 2003] allows to assert individuals like:

$$\begin{aligned} &CityCar(ka) \\ &Maker(ford) \\ &hasMaker(ka, ford) \end{aligned}$$



Domain Individual Description III

- In order to represent an individual in form of tuple through the ABox formalism, a language to express the following information is needed:
 - ▶ the name of the individual;
 - ▶ the concept to which the individual belongs;
 - ▶ the set of relations in which the individual is involved.
- Such information has to be described in accordance with the specific domain ontology.



Interaction with a Semantic Tuple Centre

- Now that we have characterised the tuple centre knowledge in terms of domain ontology and individuals, it is important to understand how this influences:
 - ▶ **tuple templates**, that are specifications of sets of tuples;
 - ▶ **tuple matching mechanism** to match templates and tuples;
 - ▶ **tuple centre primitives** to read, consume, and write tuples in the tuple centre.



Semantic Tuple Templates

- Tuple templates become specifications of set of domain individuals described by the domain ontology.
 - ⇒ A tuple template becomes a description in TBox formalism, of the set of individuals one is interested in.
 - ⇒ For example, if we are interested in the set of cars that have ford as a maker, the tuple template has to provide a concept description like $Car \sqcap (\exists \text{ hasMaker.ford})$.
 - Since a template allows to specify arguments both as input or as output, it is also necessary to specify which other kinds of individuals one is interested in.
 - ⇒ Referring to the concept $Car \sqcap (\exists \text{ hasMaker.ford})$, besides being interested in cars we could be also interested in the individual *ford*.
- ⇒ As well as the tuple case, a language is required in order to express **semantic tuple templates** in accordance with such requirements.



Semantic Tuple Matching Mechanism

- Tuple matching mechanism represents the mean whereby it is possible to identify tuples that match the specifications described by tuple templates.
- In a semantic tuple centre this means that the matching mechanism has to identify and retrieve the set of individuals specified by a TBox-based concept description, by means of the domain ontology.
- To this end, it is possible to exploit the reasoner service provided by each Description Logic-based system that allows to query the ABox through a TBox-based description and obtain the set of individuals that satisfy such query.



Semantic Tuple Centre Primitives

- In a semantic view, tuple centre primitives (in, rd, and out) represent the language whereby system components can read, consume, and write knowledge described by means of a domain ontology.
- Since the knowledge stored in the tuple centre must be always consistent with the domain ontology, there are two important consequences about such primitives:
 - ▶ in face of a writing primitive (out), it is required to check the consistency of the tuple to be written in the tuple centre, with the domain ontology. This means that, differently from the original tuple centre semantic, the out can fail in case the related tuple is not consistent with the domain ontology;
 - ▶ in order to use [inverse relations](#) (see [Baader et al., 2003]), it is needed to introduce a new primitive that allows to write set of tuples, all at once. This results from the fact that the use of the out primitive with tuples involved in an inverse relation, lead to a knowledge inconsistency.



Outline

- 1 Motivations
- 2 Semantic Tuple Centres
- 3 **TuCSon**
- 4 Semantic Tuple Centres in TuCSon
- 5 Example of ReSpecT Tuple Centre Usage
- 6 Open points
- 7 The MediCoordination Application
- 8 Distributed TBox and ABox
- 9 MSL Service Coordination via Semantic Tuple Centres
- 10 Bibliography



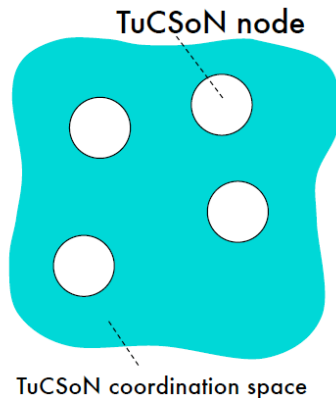
TuCSoN Coordination Model

- Tuple Centre Spread over the Network [Omicini and Zambonelli, 1999].
- It starts from the previous considerations, by proposing a notion of an associative shared dataspace whose behaviour can be tailored according to the specific application needs.
 - ▶ From tuple spaces to tuples centres [Omicini and Denti, 2001].
 - ▶ Tuple centres are programmable tuple spaces.
 - ⇒ Programmable coordination media.
 - ⇒ The coordination model it is the same.
- Tuple centres are distributed over the network, collected in nodes.
 - ⇒ Distributed coordination media.



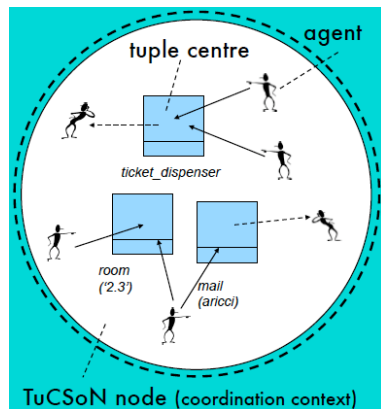
TuCSon Coordination Space

- Set of distributed nodes.
 - ▶ Each TuCSon node is an Internet node identified by the IP (logic) address.
- TuCSon **topology**.
 - ▶ Here, Internet topology.
 - ▶ **HiMAT** [Cremonini et al., 1999]: hierarchical, dynamic, configurable topology.



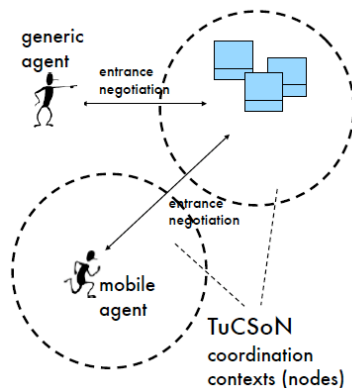
TuCSoN Node / Context I

- Each TuCSoN node defines a coordination context, providing an open / dynamic set of tuple centres as coordination media.
 - Identified by means of a logic name (term).
Ex: `mail(aricci)`,
`room('2.3')`,
`ticket_dispenser`, ...
 - Full tuple centre identifier:
`<name>@<node>`.
Ex: `mail(aricci)@myhome.org`,
`room('2.3')@ingce.unibo.it`,
`ticket_dispenser@137.204.191.188`,
...



TuCSoN Node / Context II

- In order to access and use the tuple centres of a node, an agent must enter the coordination context, either logically or physically (mobile agents).
- **Agent Coordination Context (ACC)** [Omicini, 2001].



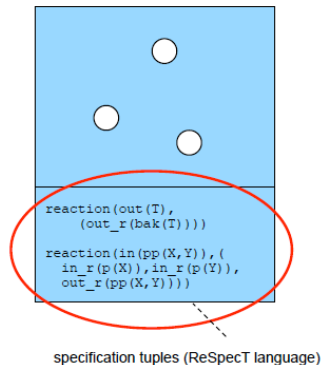
Tuple Centres Features I

- Programmable

- ▶ Tuple centre behaviour can be programmed to enact the desired coordination policies.
- ▶ **ReSpecT** is an example of programming language for specification of the behaviour.

- ★ It programs as set of **logic tuples** (**reactions**, first order logic terms such as Prolog terms) specifying medium behaviour reacting to interaction events.

⇒ Tuple centres as a general purpose coordination media customisable by means of a specification language like ReSpecT.



Tuple Centres Features II

- Adatable at runtime

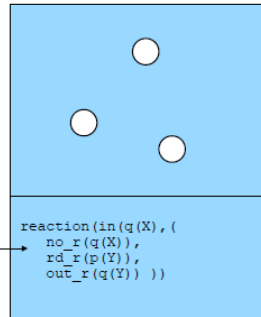
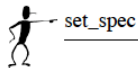
- ▶ Tuple centre behaviour can be changed / adapted dynamically, at runtime, by reprogramming the coordination media.

- Locality / encapsulation

- ▶ Tuple centres embed coordination laws.

⇒ A tuple centre can be a full-fledged coordination abstraction.

- ▶ Reaction model ensure encapsulation of low-level coordination policies.



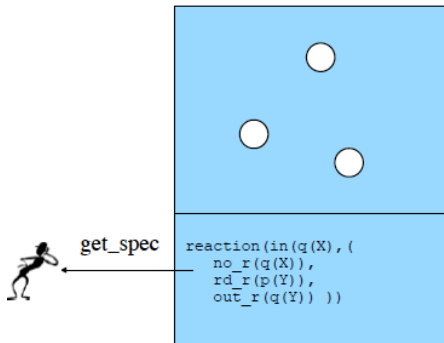
Tuple Centres Features III

- Inspectable at runtime

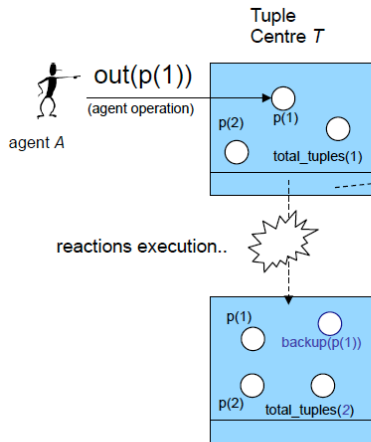
- ▶ Tuple centre behaviour can be inspected dinamically, at runtime.

- Uniformity of languages

- ▶ Same structure / primitives for communication and coordination



Simple Examples I



Current behaviour of the tuple centre (pseudocode):

When a tuple T is inserted, produce a tuple `backup(T)`

When a tuple `p(X)` is inserted, update the tuple `total_tuple(N)` (retrieve and store the tuple with `N` incremented)...

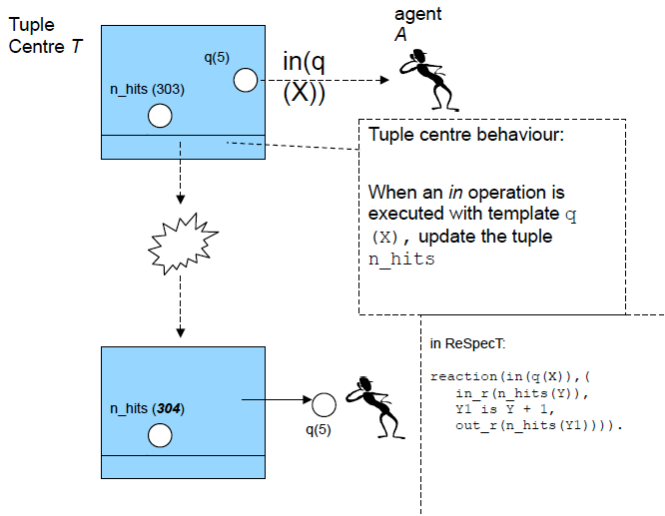
in ReSpecT:

```
reaction(out(T), (  
  out_r(backup(T)))).
```

```
reaction(out(p(X)), (  
  in_r(total_tuples(N)),  
  N1 is N + 1,  
  out_r(total_tuples(N1)))).
```



Simple Examples II



TuCSoN Technology I

• TuCSoN API

- ▶ Virtually any hosting language, currently Java and Prolog.
 - ⇒ Support for Java and Prolog agents.
- ▶ Heterogeneous hardware support.

• TuCSoN Service

- ▶ Booting the TuCSoN Service daemon.
 - ★ The host becomes a TuCSoN node.
 - ★ With current version (1.4.5):
`java -cp dir/tucson.jar alice.tucson.service.Node.`



TuCSoN Technology II

- TuCSoN Tools

- ▶ Inspector

- ★ Fundamental tool to monitor tuple centre communication and coordination state, and to debug tuple centre behaviour.
 - ★ With current version (1.4.5):
`java -cp dir/tucson.jar alice.tucson.tools.Inspector.`

- ▶ TuCSoN Shell

- ★ Shell interface for human agents.
 - ★ With current version (1.4.5):
`java -cp dir/tucson.jar alice.tucson.tools.CLIAgent.`

- TuCSoN technology is freely available in
<http://alice.unibo.it/xwiki/bin/view/TuCSoN/>



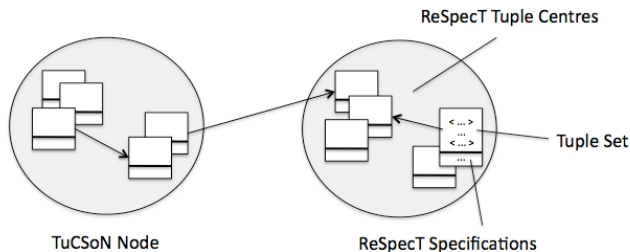
Outline

- 1 Motivations
- 2 Semantic Tuple Centres
- 3 TuCSoN
- 4 Semantic Tuple Centres in TuCSoN**
- 5 Example of ReSpecT Tuple Centre Usage
- 6 Open points
- 7 The MediCoordination Application
- 8 Distributed TBox and ABox
- 9 MSL Service Coordination via Semantic Tuple Centres
- 10 Bibliography



Tuple Centres in TuCSoN

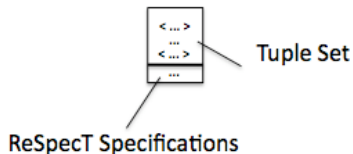
- In **TuCSoN** tuple centres are distributed over the network, collected in Internet nodes identified by an IP (logic) address.
- Tuple centres are provided by **ReSpecT** framework and they are identified by a unique name.
- Through **linkability** a tuple centre can interact with one another: a tuple centre specification can use the tuple centre primitives over other tuple centres belonging to the same or to a different node.



Semantic Tuple Centres in TuCSoN

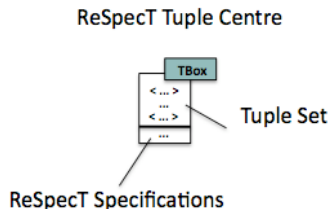
- To give a semantic to the information stored in ReSpecT tuple centres we can describe it through an ontology in terms of the concepts and relationships among them, belonging to the particular application domain.
- By adopting the [Description Logic](#) formalism we can define an ontology by means of a [TBox](#) and an [ABox](#).

ReSpecT Tuple Centre



TBox for Semantic Tuple Centres I

- TBox consists of concept descriptions, which denote sets of objects called individuals, and role descriptions, which denote binary relationships between individuals.
- Each ReSpecT tuple centre is associated to a specific TBox describing the stored information.
- In order to provide more flexibility, tuple centres in a TuCSoN node may or may not share a TBox.



TBox for Semantic Tuple Centres II

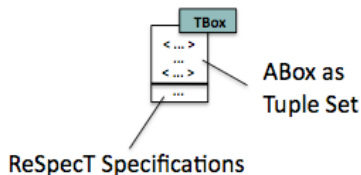
- For the TBox definition in ReSpecT tuple centres a SHOIN(D)-like description language it is exploited: **OWL-DL** ontology description language [Horrocks et al., 2003].
 - ▶ OWL is the ontology standard description language for the Semantic Web.
 - ⇒ OWL well fits the TuCSoN openness requirement.



ABox for Semantic Tuple Centres I

- ABox consists of the assertions about the individuals and roles, in terms of the terminology defined through the TBox.
- Each **tuple** stored in a tuple centre can be seen as an object belonging to the application domain, with the set of relationship in which it is involved.
 - ⇒ A tuple can be mapped into an ABox individual;
 - ⇒ The set of tuple stored in a tuple centre can be mapped into an ABox.

ReSpecT Tuple Centre



ABox for Semantic Tuple Centres II

- In order to describe tuples as ABox individuals so they can be interpreted in a semantic way by means of a TBox, we need a SHOIN(D) description language-like to specify the following information:
 - ▶ **name of the individual** we want to describe;
 - ▶ **concept** to which the individual belongs;
 - ▶ **set of roles** in which the individual is involved.
- The **semantic tuple** definition language has to preserve the tuple centre model identity.
 - ⇒ Semantic tuple structure has to be a smooth extension of the original tuple structure and semantic.



ABox for Semantic Tuple Centres III

- Below the grammar of a possible SHOIN(D)-like description language for semantic tuples is showed:

$\text{Individual} ::= I : C$

$C ::= \text{cname} \mid \text{cname}(F)$

$F ::= \text{pname} : V \mid \text{pname in } (V\text{set}) \mid F, F$

$V\text{set} ::= V \mid V, V\text{set}$

$V ::= I \mid N \mid \text{string}$

$I ::= \text{individual name}$

$N ::= \text{number}$

- Assuming to have already defined an OWL-DL TBox, by exploiting such description language we can obtain the following semantic tuple:

f550 : 'Car' (hasMaker : ferrari, hasMaxSpeed : 285, hasColour in (red, black))



Semantic Tuple Templates I

- A **tuple template** represents a specification of a set of tuples.
- In a semantic view, a tuple template can be seen as a specification of a set of domain individuals described through the domain ontology.
 - ⇒ A specification of an ABox individual set described through a TBox.
- In order to describe a tuple template according to this view, we need a SHOIN(D)-like description language allowing:
 - ▶ to express a tuple template as a description in the TBox formalism;
 - ▶ to support the specification of which kind of individuals one is interested in, since a template allows to specify arguments both as input or as output.
- The **semantic tuple template** definition language has to preserve the tuple centre model identity.
 - ⇒ Semantic tuple template structure has to be a smooth extension of the original tuple template structure and semantic.



Semantic Tuple Templates II

- Below, the grammar of a possible SHOIN(D)-like description language for semantic tuple templates is showed:

$C ::= \text{all} \mid \text{none} \mid \text{cname} \mid C, C \mid C; C \mid \text{not } C \mid R \mid \{ \text{Iset} \} \mid C(R) \mid (C)$

$R ::= F \mid \text{exists } F \mid \text{only } F \mid M$

$F ::= P \text{ in } C \mid P : I \mid P : \text{Msymb } N \mid P : = \text{string} \mid P$

$M ::= \# P : \text{Msymb } N$

$P ::= \text{pname} \mid \text{pname} / \text{vname}$

$\text{Msymb} ::= > \mid < \mid >= \mid <= \mid =$

$\text{Iset} ::= I \mid I, \text{Iset}$

$I ::= \text{individual name}$

$N ::= \text{number}$



Semantic Tuple Templates III

- By exploiting such grammar, we can define a semantic tuple template like:

`'Parent' , (only hasChild in 'Male')`

- If we are also interested in the individual belonging to the concept **Male** that has a role of kind **hasChild** with the selected individual belonging to the concept **Parent**, we can exploit the expression `pname / vname`, where `pname` and `vname` represent respectively a role name and a variable name:

`'Parent' , (exists hasChild / X in 'Male')`



Semantic Tuple Centre Primitives

- In order to assure the consistency within a semantic tuple centre with the domain ontology, a revisitation of the classical tuple centre primitives is needed.
- **in** and **rd** primitives remain unchanged since they represent query to a semantic tuple centre.
- From the other hand, **out** primitive is used to insert a semantic tuple.
 - ⇒ Before to insert a semantic tuple it is necessary to check the consistency of such tuple with the ontology associated to the semantic tuple centre.
 - ⇒ Differently from the original tuple centre model, out primitive of a semantic tuple may fail.



Semantic Tuple Matching Mechanism

- Tuple matching mechanism represents the mean by which it is possible to identify tuples that match the specification described by a tuple template.
- ⇒ **Semantic tuple matching mechanism** represents the mean by which it is possible to identify individuals that match the specifications described by a concept specification.
- ReSpecT tuple centres where tuples and templates are represented as first order logic terms and the tuple matching mechanism adopted is logic unification.
- ⇒ An extension of the ReSpecT tuple matching mechanism it is needed.



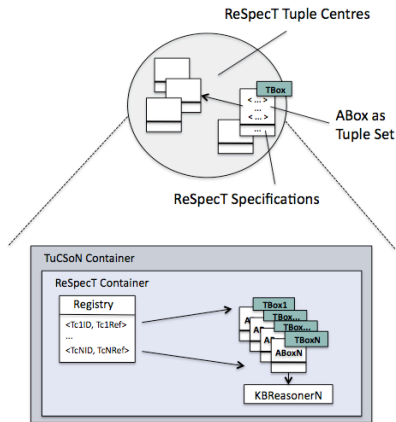
Interactions with ReSpecT Semantic Tuple Centres

- In order to make work semantic tuple centres in ReSpecT framework it is needed:
 - ▶ a mechanism to relate a domain ontology to each tuple centre;
 - ▶ a mechanism to interpret semantic tuples and semantic tuple templates described with the respective SHOIN(D)-like description language;
 - ▶ a SHOIN(D) reasoner to both check semantic tuple consistency and realise the semantic tuple matching mechanism.
- Besides semantic tuples, tuple templates, and tuple matching mechanism, in a semantic tuple centre it is useful to preserve the possibility of using no semantic ones.
 - ⇒ No semantic tuples, tuple templates, and tuple matching mechanism are useful to realise coordination mechanisms in tuple centres.



An Extension of ReSpecT Framework I

- Tuple centres are provided in each TuCSoN node by the ReSpecT **container**.
 - The container represents the manager of the tuple centre life-cycle and provides the API to create, access, and use them.
- ⇒ The container has to provide the API to:
- ▶ create a tuple centre associated with a specified TBox and a reasoner;
 - ▶ set and obtain the TBox related to a particular tuple centre.



An Extension of ReSpecT Framework II

- In this work we adopted the OWL reasoner [Pellet](#) [Sirin et al., 2007] because:
 - ▶ is easy to integrate it with ReSpecT since it is open-source and it is written in java;
 - ▶ is a complete and capable OWL-DL reasoner with good performance.
 - ▶ does the [Open World Assumption](#) (OWA) that is a desirable feature since in a open and dynamic environment information can be incomplete.
- When the container create a new tuple centre:
 - ▶ by exploiting the Pellet API, a new instance of ontology is created from a specified OWL file and provided to the tuple centre;
 - ▶ a new instance of the Pellet reasoner is created and provided to the tuple centre.



An Extension of ReSpecT Framework III

- In face of an **out** primitive a tuple centre:
 - ▶ interpreters the individual expressed by the received semantic tuple;
 - ▶ by exploiting the Pellet reasoner, checks the individual consistency with the ontology;
 - ▶ inserts the individual in the ABox.
- In face of a **in** or **rd** primitive a tuple centre:
 - ▶ interpreters the concept specification expressed by the received semantic tuple template;
 - ▶ converts the tuple template in **SPARQL** language;
 - ▶ through the API provided by Pellet, queries the Pellet reasoner in order to obtain the individuals identified by the SPARQL specification;
 - ▶ converts the first individual obtained by the reasoner in a semantic tuple;
 - ▶ returns the produced semantic tuple to the requester.



Outline

- 1 Motivations
- 2 Semantic Tuple Centres
- 3 TuCSoN
- 4 Semantic Tuple Centres in TuCSoN
- 5 Example of ReSpecT Tuple Centre Usage**
- 6 Open points
- 7 The MediCoordination Application
- 8 Distributed TBox and ABox
- 9 MSL Service Coordination via Semantic Tuple Centres
- 10 Bibliography



Scenario

- A pervasive application provided as a service to customers of a [shop for second-hand cars](#), equipped with a mobile device such as a PDA or a mobile phone.
 - ▶ When the customer enters into the car shop, by means of a mobile device it may compose a car pattern describing preferences and communicate them to a software agent that runs in the device.
 - ▶ Every time a customer enters into a shop room, a display shows the set of cars compatible with her preferences as well as the car position in a room map.



Possible Problems I

- In an open and dynamic context, we can not assume the customer agent would have a priori knowledge about the information provided by the shop service, neither about the syntactic structure of such information.
 - ⇒ The software agent would not be able to coordinate itself with the shop application in order to communicate the customer preferences and provide the customer with the display service.
 - ⇒ In order to face this problem we can exploit semantic techniques.



Possible Problems II

- In an open and dynamic context, it is also important to uncouple application coordinables, that are the customer agent and the shop service.
 - ⇒ It is necessary to specify the coordination laws managing the interaction between the customer agent and a room display, outside the agent itself.
 - ⇒ The task of the customer agent should be only to acquire and reason upon the domain ontology and send the customer preferences to the shop services without taking care of the coordination laws allowing the visualisation of the set of cars compatible with the user preferences.
 - ⇒ In order to face this problem we can exploit the tuple centre coordination model.



A Possible Solution I

- We can define an ontology describing the application domain concepts as well as the relationships among such concepts.
- Such ontology can be acquired by the customer agent in a dynamic way, when it enters into the shop building.
- Since the ontology is expressed by means of a DL-based language, the agent can reason upon such ontology, shape a smart interface by which the customer can communicate its car preferences and send them to the shop service.



A Possible Solution II

- We can define a set of ReSpecT semantic tuple centres spread over several TuCSoN nodes:
 - ▶ A central node containing a tuple centre storing the customer agent identity and customer car preferences;
 - ▶ A node for each room containing the descriptions of the cars available in such room.



A Possible Solution III

- When the customer enters into the shop building, the agent running in its device writes into the central-node tuple centre the following tuple containing the user identity and car preferences:

```
user(userId, 'Car'(hasMaxSpeed:>160; hasMaker in renault, ford))
```

- When the customer enters into a new room, a sensor detects its mobile device and writes the following tuple containing the user identity into the room-node tuple centre:

```
entered_user(userId)
```



A Possible Solution IV

- Let's suppose the room-node tuple centre contains the following tuples describing the cars that are in the room:

```
ka: 'CityCar' ( hasMaker=ford,  
                hasMaxSpeed=165,  
                hasPrice=8000)  
mondeo: 'FamilyCar' ( hasMaker=ford,  
                      hasMaxSpeed=195,  
                      hasPrice=11000)  
twingo: 'CompactCar' (hasMaker=renault,  
                      hasMaxSpeed=150,  
                      hasPrice=6000)
```



A Possible Solution V

- Moreover, let's suppose the tuple centre also contains the following reaction:

```
reaction(out(entered_user(UId)),  
         (operation, invocation)  
         ( central-node_tc@central-node_address ?  
           rd(user(UId, Preferences)),  
           in(rd-all(CarList matching Preferences)),  
           write_display(CarList))  
         ).
```



A Possible Solution VI

- By means of such tuple centre reaction, when in a room-node tuple centre a tuple like `entered_user(userId)` is written, the following operations are executed:
 - ▶ the variable `UId` is unified with the user identifier `userId`;
 - ▶ the tuple `user(userId, 'Car'(hasMaxSpeed:>160; hasMaker in {renault, ford}))` is read from the central-node tuple centre and the variable `Preferences` is unified with the concept description `user(userId, 'Car'(hasMaxSpeed:>160; hasMaker in {renault, ford}))`;
 - ▶ the tuple `user(userId, 'Car'(hasMaxSpeed:>160; hasMaker in {renault, ford}))` is read from the central-node tuple centre;



A Possible Solution VII

- By means of such tuple centre reaction, when in a room-node tuple centre a tuple like `entered_user(userId)` is written, the following operations are executed:
 - ▶ the following set of cars, denoted by the semantic tuple template `'Car'(hasMaxSpeed:>160; hasMaker in renault, ford)`
 - ▶ is read and unified with the variable `CarList`:

```
ka:'CityCar'(  
  hasMaker=ford,hasMaxSpeed=165,hasPrice=8000)  
  
mondeo:'FamilyCar'(  
  hasMaker=ford,hasMaxSpeed=195,hasPrice=11000)
```
 - ▶ the car set identified is then sent to the room display through the command `write_display`.



Outline

- 1 Motivations
- 2 Semantic Tuple Centres
- 3 TuCSoN
- 4 Semantic Tuple Centres in TuCSoN
- 5 Example of ReSpecT Tuple Centre Usage
- 6 Open points**
- 7 The MediCoordination Application
- 8 Distributed TBox and ABox
- 9 MSL Service Coordination via Semantic Tuple Centres
- 10 Bibliography



Main Open Points

- Is a tuple an individual or is a term an individual?
- A SHOIN(D)-like definition language for TBox.
- Run-time modification of TBoxes.
- Distributed ABox / TBox.



Outline

- 1 Motivations
- 2 Semantic Tuple Centres
- 3 TuCSoN
- 4 Semantic Tuple Centres in TuCSoN
- 5 Example of ReSpecT Tuple Centre Usage
- 6 Open points
- 7 The MediCoordination Application**
- 8 Distributed TBox and ABox
- 9 MSL Service Coordination via Semantic Tuple Centres
- 10 Bibliography



Aim

- In medical domain, unique documents are considered part of a single common document related to a single patient.
- The complete set of documents, called **fragments**, is named **Patient Electronic Health Record** (PEHR).
- **MediCoordination objective**: Provide a totally decentralised storage for PEHRs through a federated metadata infrastructure allowing semantic descriptions of medical documents thus allowing a better integration and interoperability between heterogeneous environments.



Fragments and Metadata I

- Linked to some other fragments in order to form a PEHR.
- Associated to particular access rights, which allow or prevent medical personnel to read or modify fragments, patient, and medical professional through [metadata annotations](#).
- A metadata annotation is described as a three-field RDF structure called triple:
 - ▶ [Fragment id](#) represented by an unique URL;
 - ▶ [Attribute-value](#) representing a fragment property.



Fragments and Metadata II

- Metadata annotations are stored independently from the documents they annotate.
- Metadata repositories are disseminated among care institutions (local) in a federated structure.
- The federation is a network mesh where each node is connected to its neighbours.



SACoord Infrastructure

- SACoord is used to implement the **Metadata Service Layer** (MSL) component of the MediCoordination Platform, whose goal is to manage **federated metadata** about medical documents.
- SACoord coordination infrastructure exploits a modified version of **WSDir**.
- WSDir is a framework for building a federation of semantic web services.
 - ▶ Allows registration and discovery of semantic web services;
 - ▶ Provides storage and discovery mechanisms.



Why Use Semantic Tuple Centres Instead of WSDir approach?

- WSDir approach couples MSL services.
- By exploiting the tuple centre coordination model it is possible to:
 - ▶ decouple MSL services in name, space, and time;
 - ▶ put coordination laws outside from MSL services by encapsulating such laws in tuple centre specifications;
 - ▶ through ReSpecT tuple centre, exploit intelligent agents in order to realise automated behaviours.

⇒ Tuple centre model provide design advantages leading to application reuse and openness.



Outline

- 1 Motivations
- 2 Semantic Tuple Centres
- 3 TuCSoN
- 4 Semantic Tuple Centres in TuCSoN
- 5 Example of ReSpecT Tuple Centre Usage
- 6 Open points
- 7 The MediCoordination Application
- 8 Distributed TBox and ABox**
- 9 MSL Service Coordination via Semantic Tuple Centres
- 10 Bibliography



A Tableau-based Federated Reasoning Algorithm for Modular Ontologies [Bao et al., 2006]

- Is based on **Package-based Description Logics** (P-DL).
- In P-DL an ontology is composed of a collection of modules called **packages**.
- Each name of concept, property, and individual or axiom is associated with a **home package**.
- A package can use terms defined in other packages.
- A P-DL ontology is able to model inter-module concept subsumption and role relations.
- The algorithm check the satisfiability of a concept C w.r.t. a distributed DL Ontology TBox .



Distributed Instance Retrieval in Heterogeneous Ontologies I [Luciano et al., 2005]

- Based on Distributed Description Logic [Borgida and Serafini, 2002] where the semantic mapping between ontologies are expressed as **bridge rules** between pairs of concepts belonging to different ontologies.
- The semantic correspondence between heterogeneous individuals belonging to different ABoxes are expressed as **individual correspondence**.
- A DDL is not able to model inter-role relations neither cyclic references of concepts among ontology modules.



Distributed Instance Retrieval in Heterogeneous Ontologies II [Luciano et al., 2005]

- Provide an algorithm for the distributed instance retrieval of a particular concept.
- The algorithm it was implemented on top of **DRAGO** [Serafini and Tamin, 2005], a distributed reasoning system that organises a set of reasoning peers into a peer-to-peer network.



A Decentralised and Ontology-Based Approach to Infrastructure Monitoring I [Fuchs and Berger, 2008]

- A resoner layer realised in a decentralised manner with reasoning services located nearby each repository and collaborating for performing reasoning tasks.
- Distributed knowledge base (kb) with a common TBox, whereas the ABox is split in to N ABoxes, one for each kb.
- For every ABox instance a , which is mentioned in more than one ABox, it holds that a is asserted to be a member of only the special concept C disjoint from all other concepts and not used in the definition of other concepts.



A Decentralised and Ontology-Based Approach to Infrastructure Monitoring II [Fuchs and Berger, 2008]

- Associated with each reasoning service is Profile(R), which represents an abstract of the knowledge that can be provided by R as a means for optimising query answering.
- There is also a directory service which indexes Profile(R)s.
- The query to be answered are grounded conjunctive query, where each query can be a concept query atom or a property query atom.
- The query execution plan is constructed conjunct by conjunct and for each conjunct, the relevant reasoning services are identified by their Profile(R) stored in the directory service.
- The same approach is used in [An architecture for peer-to-peer reasoning](#) [Anadiotis et al., 2007].



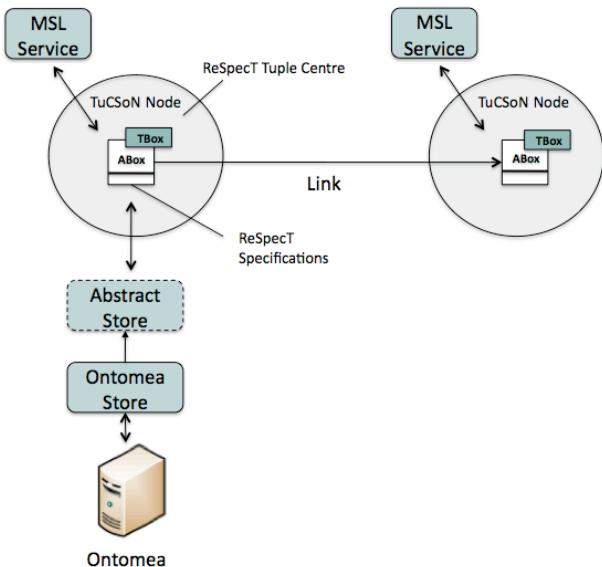
Outline

- 1 Motivations
- 2 Semantic Tuple Centres
- 3 TuCSoN
- 4 Semantic Tuple Centres in TuCSoN
- 5 Example of ReSpecT Tuple Centre Usage
- 6 Open points
- 7 The MediCoordination Application
- 8 Distributed TBox and ABox
- 9 MSL Service Coordination via Semantic Tuple Centres
- 10 Bibliography



MSL Service Coordination

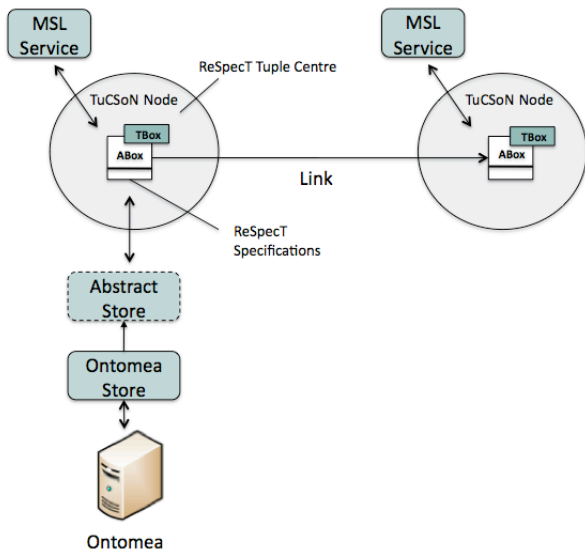
- A MSL service provides:
 - ▶ a web service interface to the external world;
 - ▶ web service security technologies.
- TuCSoN infrastructure supports semantic coordination among MSL services:
 - ▶ in a decoupled way;
 - ▶ based upon policies that restrict the access to fragment stores.



Distributed Abox

- Starting from [Luciano et al., 2005] and [Serafini and Tamilin, 2005] or [Fuchs and Berger, 2008]:

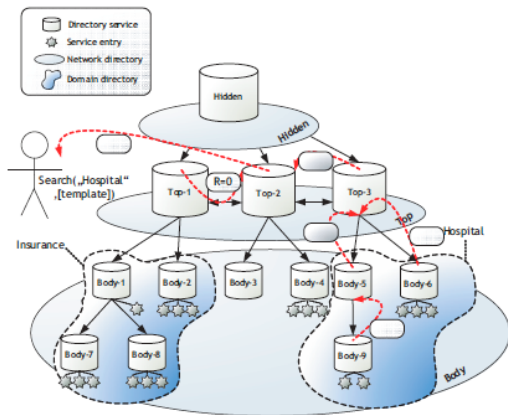
- ▶ exploit tuple centre programmability and linkability to coordinate reasoners.



Semantic Self-organising Coordination I

- Starting from the deliverable D2.4 of [Triple Space Computing](#) project and the [Coordination through Chemical-inspired Tuple Spaces](#) work:

- exploit tuple centre programmability and linkability in order to introduce semantic self-organising coordination laws
- to automatically aggregate data on the basis of an [emergent semantic similarity function](#) in order to improve the semantic-data-search efficiency.



Semantic Self-organising Coordination II

- For example we can use user requests in order to make semantic similarity functions to be associated to each node.
- When a similarity function emerges, through diffusion, competition, and aggregation operation provided by the chemical-based-coordination framework, we can move semantic data respecting such function, in the related node.
- Semantic similarity functions will be used to select and cut tree branches during the data searching.



Steps

- Sicurezza: RBAC e pensare ai diversi livelli di sicurezza e a come integrarli in TuCSoN
- Integrazione fra TuCSoN e Triple Store
- Topologia: come posso propagare le query?
- Sicurezza: trust domain for self-org coord
- Self-org: real-time emergency
- Self-org: emergent semantic similarity function con esempio



Outline

- 1 Motivations
- 2 Semantic Tuple Centres
- 3 TuCSoN
- 4 Semantic Tuple Centres in TuCSoN
- 5 Example of ReSpecT Tuple Centre Usage
- 6 Open points
- 7 The MediCoordination Application
- 8 Distributed TBox and ABox
- 9 MSL Service Coordination via Semantic Tuple Centres
- 10 Bibliography**



Bibliography I



Anadiotis, G., Kotoulas, S., and Siebes, R. (2007).

An architecture for p2p reasoning.

In *New forms of reasoning for the Semantic Web: scalable, tolerant and dynamic*.



Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003).

The Description Logic Handbook: Theory, Implementation, and Applications.

Cambridge University Press.



Bandara, A., Payne, T., Roure, D. D., Gibbins, N., and Lewis, T. (2008).

Semantic resource matching for pervasive environments: The approach and its evaluation.

Technical Report ECSTR-IAM08-001, Southampton, UK.



Bibliography II

-  Bao, J., Caragea, D., and Honavar, V. (2006).

A tableau-based federated reasoning algorithm for modular ontologies.
In *WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 404–410, Washington, DC, USA. IEEE Computer Society.

-  Borgida, A. and Serafini, L. (2002).

Distributed description logics: Directed domain correspondences in federated information sources.

In *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002*, pages 36–53, London, UK. Springer-Verlag.

-  Ciancarini, P. (1996).

Coordination models and languages as software integrators.
ACM Comput. Surv., 28(2):300–302.



Bibliography III



Cremonini, M., Omicini, A., and Zambonelli, F. (1999).

Multi-agent systems on the Internet: Extending the scope of coordination towards security and topology.

In Garijo, F. J. and Boman, M., editors, *Multi-Agent Systems Engineering*, volume 1647 of *LNAI*, pages 77–88. Springer-Verlag. 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99), Valencia, Spain, 30 June–2 July 1999. Proceedings.



Fuchs, F. and Berger, M. (2008).

A decentralized and ontology-based approach to infrastructure monitoring.

In *Multikonferenz Wirtschaftsinformatik*.



Horrocks, I., Patel-Schneider, P. F., and Harmelen, F. V. (2003).

From shiq and rdf to owl: The making of a web ontology language. *Journal of Web Semantics*, 1:2003.



Bibliography IV

-  Luciano, Serafini, L., and Tamilin, A. (2005).

Distributed instance retrieval in heterogeneous ontologies.
In In Proceedings of SWAP 2005, CEUR Workshop Vol 166.

-  Nixon, L. j. b., Simperl, E., Krummenacher, R., and Martin-recuerda, F. (2008).

Tuplespace-based computing for the semantic web: A survey of the state-of-the-art.
Knowl. Eng. Rev., 23(2):181–212.

-  Omicini, A. (1999).

On the semantics of tuple-based coordination models.
In 1999 ACM Symposium on Applied Computing (SAC'99), pages 175–182, San Antonio, TX, USA. ACM.

-  Omicini, A. (2001).

On the notion of agent coordination context: Preliminary notes.
*AI*IA Notizie*, XIV(4):44–46.



Bibliography V



Omicini, A. and Denti, E. (2001).

From tuple spaces to tuple centres.

Science of Computer Programming, 41(3):277–294.



Omicini, A. and Zambonelli, F. (1999).

Coordination for Internet application development.

Autonomous Agents and Multi-Agent Systems, 2(3):251–269.

Special Issue: Coordination Mechanisms for Web Agents.



Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K. P. (2002).

Semantic matching of web services capabilities.

In *ISWC '02: Proc. of the First Int'l Semantic Web Conference on The Semantic Web*, pages 333–347, London, UK. Springer.



Rossi, D., Cabri, G., and Denti, E. (2001).

Tuple-based technologies for coordination.

pages 83–109.



Bibliography VI



Serafini, L. and Tamin, A. (2005).

Drago: Distributed reasoning architecture for the semantic web.

In *In ESWC*, pages 361–376. Springer.



Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. (2007).

Pellet: A practical owl-dl reasoner.

J. Web Sem., 5(2):51–53.



Coordination in Open and Dynamic Environments via TuCSoN Semantic Tuple Centres

Elena Nardini

ALMA MATER STUDIORUM – Università di Bologna
`elena.nardini@unibo.it`

December 4, 2009

