Towards Scalability in Tuple Spaces

Philipp Obreiter, Guntram Gräf
Telecooperation Office (TecO), University of Karlsruhe,
Vincenz-Prießnitz Str. 1, 76131 Karlsruhe, GERMANY
Phone: +49 (721) 6902-79, Fax: +49 (721) 6902-16
{obreiter, graef}@teco.uni-karlsruhe.de

ABSTRACT

Applications in eCommerce and Ubiquitous Computing ask for coordination of highly distributed and heterogenous data sources and services. Tuple spaces offer a data-driven coordination model, hence they may be used for this purpose. However, research on distributed tuple spaces has not resolved yet how to render tuple spaces scalable. This is partly due to their informal conception. This paper formalizes tuple spaces and introduces a new concept for achieving scalability. It generalizes existing concepts and may lead to scalability in some application areas.

Keywords

Tuple Spaces, scalability, formalization, distribution

1. INTRODUCTION

Applications in the emerging fields of eCommerce [6] and Ubiquitous Computing [15] are composed of heterogenous systems that have been designed separately. Hence, these systems are loosely coupled and require a coordination mechanism that is able to gap spatial and temporal remoteness. The use of tuple spaces [8] for data-driven coordination of these systems has been proposed in the past [7]. In addition, applications of eCommerce and Ubiquitous Computing are not bound to a predefined size, so that the underlying coordination mechansim has to be highly scalable. However, it seems to be difficult to conceive a scalable tuple space.

The paper is organized as follows. Chapter 2 gives an overview of existing approaches for achieving scalability in tuple spaces and their shortcomings are pointed out. Chapter 3 and 4 formalize tuple spaces and scalability respectively. A proposed concept based on hypercubes is discussed in chapter 5.

2. STATE OF THE ART

2.1 The Original Concept and its Extensions

A *tuple space* [8] is a logically shared associative memory that enables cooperation based on the blackboard design pattern [11]. *Tuples* may be written to the tuple space and they are retrieved as specified by *templates*. Tuples and templates are ordered collections of *fields* that can be either *actual* or *formal*. An actual field has a specific value, whereas a formal field represents a set of values. There is no schematic restriction on how fields are composed to tuples and templates. A reading operation returns a tuple that is matched by a template. Matching is the key concept of tuple spaces, because it enables associative yet only partly specified retrieval of tuples.

Several extensions of this concept have been proposed in the past [2], [7], [9], [16]. E.g. *object orientation* has been introduced to tuple spaces [2] and [7] suggests the use of *semantic* templates that match tuples structurally. There are several implementations of tuple spaces, e.g. Linda [8], JavaSpaces [13] and T Spaces [17]. They differ in the amount of extensions implemented.

2.2 Prior Studies of Scalability

A scalable tuple space is inherently distributed. Different concepts for distributing tuples have been suggested in the past. However, remarkably few of them aim at scalability.

In [12], an adaptive mechanism is set in place that automatically moves tuples to the server with the lowest cost. E.g. if an application exclusively uses specific tuples, they are moved to the server nearest to the application. Therefore, this concept improves performance, if access to tuples comes with locality of space and time. However, some applications make use of a tuple space, in order to gap space or time remoteness. Hence, this mechanism may lead to performance gains in some application areas, but it is no general concept for scalability. Yet another approach [4] includes replication of tuples and induces a logical structure on the servers. It is assumed that cooperating applications are logically near. However, such an assumption may be correct in parallel processing, but not for other applications of tuple spaces. Furthermore, this concept is not really scalable, because some servers become bottlenecks due to the logical structure. In addition, it is difficult to dynamically adjust the number of servers.

All of these concepts strictly rely upon locality of access and thus they regard tuples as black boxes. Since locality cannot not be assumed in general, another approach [6] distributes tuples based on a tuple's attributes. Hence, retrieval of tuples is performed on servers that are determined by the template's attributes. However, templates do not have to fully specify the attributes of the tuples that they match. Therefore, it is necessary to identify attributes that are shared by a template and the tuples matched.

The use of hash functions has been suggested [1], [10] for this purpose. According to its hash code, a tuple or template is distributed to either an arbitrary server or to all servers. Hence, the concept of hash functions lacks a fine granular distribution strategy. Furthermore, it relies on the proliferation of an appropriate hash function by the application programmer, if scalability is to be achieved. In most application areas, this is a highly non trivial task that, in addition, often is not solvable. In

SAC2002, Madrid, Spain

spite of that, this contribution introduces a new concept that is a refinement of this approach.

3. ANALYSIS

In order to achieve scalability, structural restrictions of the scheme have to be exploited. E.g. in relational databases, the uniqueness of primary key values is used. However, the structure of tuple spaces as introduced in [8] has recently been extended by object orientation and semantic tuples [7]. As a result, tuple spaces are more expressive, but important structural restrictions are set aside. E.g. in JavaSpaces [13] and TSpaces [17] matching of a tuple can be implemented regardless of its structure, i.e. its fields. Therefore, a formalization of tuple spaces must take into account different levels of expressiveness.

3.1 Formalization of Tuple Spaces

In a first step, fields and tuples are formalized in a way, that integrates extentions. One key concept is to regard templates as tuples [7], so that matching induces a structure on tuples and fields. In the following, the term *template* depicts a tuple with a certain role, i.e. the specification of a reading access.

Actual and formal tuples are introduced as vectors of fields. In addition, semantic tuples are defined as sets of actual and formal tuples.

Fields. Let C denote the set of classes and let I_c denote the set of instances of $c \in C$, with $c \neq c'$ implying $I_c \cap I_{c'} = \emptyset$. The classes are ordered by $\leq_c \subseteq C^2$, with $c \leq_c c'$ if and only if c is c' or c is a superclass of c'. Multi-inheritence is explicitly allowed, but $\leq_c$ has to be antisymmetric. Therefore, $(C, \leq_c)$ is a *partially ordered* set. It is assumed that there exists a *minimal element* $\perp_F \in C$, i.e. $\perp_F \leq_c c$ for all $c \in C$. E.g. in Java [14] $\perp_F$ is the class `object`. Let I denote the set of all instances. Elements of C are called *formal fields* and elements of I are called *actual fields*. Therefore, the set of fields F is defined as

$$F = C \cup \bigcup_{c \in C} I_c .$$

Let class: $F \rightarrow C$ denote the mapping $class(c) = c = class(i)$ for any $c \in C$ and $i \in I_c$. $\leq_c$ partly implies matching on fields, because c matches c' if and only if $c \leq_c c'$. Furthermore, an actual field $i \in I_c$ has to be matched by every superclass of c. Therefore, $match_F \subseteq F^2$ is a *matching relation* on F if and only if

$$\forall c \in C: \forall f \in F: c \leq_c class(f) \leftrightarrow match_F(c,f) .$$

Hence, $match_F \cap C^2 = \leq_c$. This definition of matching imposes no restriction on matching between actual tuples. E.g. in [13], [17] matching is freely customizable by polymorphic matching methods.

Tuples. Let $\tau_{formal}(F)$ and $\tau_{actual}(F)$ denote the set of *formal* and *actual tuples* to a given set of Fields F. If the dimension of formal and actual tuples is not limited to a maximal dimension d, d is set to ∞. In addition, $\tau(F)$ is defined as the set of formal and actual tuples by

$$\tau(F) = \bigcup_{i=1}^{d} F^i , \quad \tau_{actual}(F) = \bigcup_{i=1}^{d} I^i , \quad \tau_{formal}(F) = \tau(F) \setminus \tau_{actual}(F) .$$

Let $\Gamma(F)$ denote the set of *semantic* tuples with

$$\perp_{\Im(F)} = \bigcup_{i=1}^{d} \{\perp_F\}^i \subseteq P(\tau(F)) , \quad \perp_{\Im(F)} \in \Gamma(F) \subseteq P(\tau(F)) \setminus \{\emptyset\}$$

with $P(A)$ depicting the power set [3] of A. Then $\Im(F) := \Gamma(F) \cup \tau(F)$ is called the *set of tuples*. It depends on F, but an explicit depiction of this dependency may be omitted in later chapters by using $\Im$, Γ and τ. There is at least one semantic tuple in $\Im(F)$, i.e. $\perp_{\Im(F)}$. Furthermore, let $\Pi_i: \tau(F) \rightarrow F$ denote the *projection* of a vector in $\tau(F)$ to the field at position i.

Matching. Let $match_\Im$ denote the *matching relation* on tuples. In order to be as expressive as in [13], [17], no restriction for the matching on tuples is applied, except of $match_\Im \subseteq \Im(F)^2$.

Tuple Space Schemes. Let F, $match_F$, $\Im(F)$ and $match_\Im$ denote sets that comply with above restrictions. Then, the quadruple $(F, match_F, \Im(F), match_\Im)$ is called a *tuple space scheme*. Ψ is defined as the set of tuple space schemes.

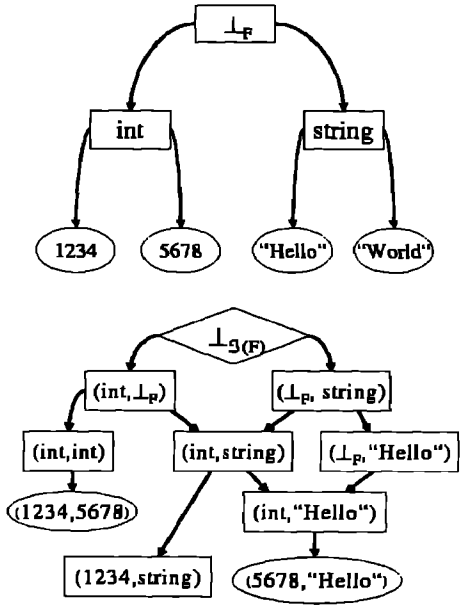

Figure 1. Excerpt of an exemplary tuple space scheme that can be used in Linda. $(F, match_F)$ is shown above and $(\Im(F), match_\Im)$ is shown below.

Example. F and $\Im(F)$ are sets ordered by $match_F$ and $match_\Im$. Therefore, $(F, match_F)$ and $(\Im(F), match_\Im)$ can be visualized as graphs [3]. Semantic, formal and actual fields or tuples are represented as rhombi, rectangles and circles. In the following, reflexivity and transitivity is obmitted in the figures, if obvious from the context. Furthermore, only parts of the graphs are shown, because generally F and $\Im(F)$ are infinite. Figure 1 shows an example of a scheme.

Alternatively, tuples may be visualized based on the graph $(\tau(F), match_\Im)$. Then, semantic tuples are represented as hypergraphs [3].

3.2 The Subset Ψ^* of Tuple Space Schemes

Tuple space schemes are very expressive. Except for semantic tuples and multiple inheritance, JavaSpaces [13] and T Spaces [17] allow such schemes to be implemented. However, Linda [8] does not support object orientation, user defined field matching, semantic templates and user defined tuple matching.

Therefore, the following sections are confined to the subset $\Psi^* \subseteq \Psi$ of quadruples $(F,match_F,\Im(F),match_\Im)$ that comply with:

a) $\leq_F := match_F$ is an order and the *infimum* on F is well-defined. Therefore, $(F,\leq_F)$ is a semilattice.

b) The only semantic tuple is $\perp_{\Im(F)}$.

c) Matching of tuples is performed by matching the fields of a template to the one of a tuple [8]. Hence, $\leq_\Im := match_\Im$ is an order with

$$\leq_\Im := \{\ (\perp_{\Im(F)}) \times \Im(F)\ \} \cup \{\ (t_1,t_2) \in \tau(F)^2 \mid |t_1|=|t_2| \wedge \forall i \in [1,\ldots,|t_1|]: match_F(\Pi_i(t_1),\Pi_i(t_2))\ \}\ .$$

4. FORMALIZATION OF SCALABILITY

This work is focused on how tuple spaces can scale up with the stored tuples and their retrieval. However, resources on a single tuple space server are limited. Therefore, tuples have to be distributed on several servers, in order to achieve scalability. Former approaches to scalability [1], [10], [12] have different mechanisms on how tuples and templates are assigned to servers. Note, that it does not suffice to achieve scalability of the total size of tuples stored. E.g. an approach is not scalable, if matching on an arbitrary template is done by querying every server.

4.1 Distribution

Let p denote the number of servers that store tuples. Furthermore, it is supposed that the servers are indexed from *1* to p. In the following, a server is identified by its index. Therefore, the servers are represented by the set $\{1,\ldots,p\}$. In addition, $\Im$ depicts a set of tuples, as defined in chapter 3. Let Δ denote the set of mappings $\Im \rightarrow P(\{1,\ldots,p\}) \setminus \varnothing$, called distributions.

Definition. $\delta \in \Delta$ is a *permissible distribution* if and only if

$$\forall T_1,T_2 \in \Im: match_\Im(T_1,T_2) \rightarrow \delta(T_1) \cap \delta(T_2) \neq \varnothing\ .$$

Δ_p denotes the set of permissible distributions. They ensure that matching tuples share a common server. If every tuple T_1 is stored to $\delta(T_1)$, then it is enough to confine to $\delta(T_2)$, in order to find tuples matched by T_2.

Example. For an arbitrary hash function $h: \Im \rightarrow Nat$, [6] suggests the distribution δ_h

$$\delta_h(T) := \{\ 1 + [h(T) \bmod p]\ \}\ .$$

However, δ_h is not a permissible distribution, because $\delta_h(\perp_\Im) \neq \{1,\ldots,p\}$.

Permissible distributions do not distinguish tuples from templates, although a distinction based on the role of a tuple could be reasonable.

Definition. With $\delta_w,\delta_r \in \Delta$, (δ_w,δ_r) is a *permissible write/read distribution* if and only if

$$\forall T_1,T_2 \in \Im: match_\Im(T_1,T_2) \rightarrow \delta_r(T_1) \cap \delta_w(T_2) \neq \varnothing\ .$$

Let Δ_{wr} denote the set of permissible write/read distributions. Δ_{wr} is not empty, since $\delta \in \Delta_p$ implies $(\delta,\delta) \in \Delta_{wr}$. Δ_{wr} can be regarded as the asymmetric extention of Δ_p. Semantically, a tuple T_1 that is to be written to the tuple space, is stored to $\delta_w(T_1)$. Then, a reading access with the template T_2 may be confined to $\delta_r(T_2)$. Note, that the cardinality of $\delta_w(T)$ does not have to be one. Hence, this formalism does not impose any restriction on the replication of tuples among several servers.

Example. Let $\delta_1,\delta^* \in \Delta$ with

$$\forall T \in \Im: |\delta_1(T)|=1 \wedge |\delta^*(T)|=p\ .$$

Then, (δ^*,δ_1) and (δ_1,δ^*) are both permissible write/read distributions. The strategy pursued by (δ^*,δ_1), is to write tuples to every server, so that retrieving tuples is confined to an arbitrary server. On the contrary, (δ_1,δ^*) implies that tuples are only written to one server, hence every server has to be queried for retrieval. Figure 2 illustrates this principle.

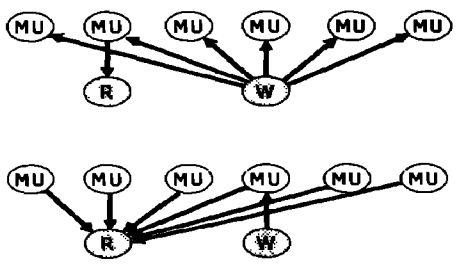

Figure 2. $(\delta^*,\delta_1) \in \Delta_{wr}$ *(above)* and $(\delta_1,\delta^*) \in \Delta_{wr}$ *(below)*. The arrows indicate which servers *(MU)* are taken into account for writing *(W)* and reading *(R)* a tuple.

4.2 A Deterministic Model

Let $\pi_w,\pi_r: \Im \rightarrow [0,1]$ denote the mapping of tuples to their *frequency of use* in write and read operations. Therefore, Π_{wr} is defined as the set of *usage profiles* (π_w,π_r) with

$$\sum_{T \in \Im} \pi_w(T) = 1 = \sum_{T \in \Im} \pi_r(T)\ .$$

In this section, a determistic model is introduced which describes static and dynamic behaviour of a tuple space. The model is based on a tuple space scheme $(F,match_F,\Im(F),match_\Im) \in \Psi$, a permissible write/read distribution $(\delta_w,\delta_r) \in \Delta_{wr}$ and a usage profile $(\pi_w,\pi_r) \in \Pi_{wr}$.

Let $\Im_n \subseteq \Im$ denote a multiset of n tuples, that is the tuple space. $\Im_n(q) \subseteq \Im_n$ is defined as the multiset of tuples on server q by

$$\Im_n(q) := \{\ T \in \Im_n \mid q \in \delta_w(T)\ \}\ .$$

Let $S_M(q)$ and $S_Q(q)$ denote the resources needed on server q for *storing tuples* and for *processing queries* respectively. A processing query is a test on whether a server contains a tuple that is matched by a template. The unit of S_M is tuples, hence this models abstracts from the size of tuples. The unit of S_Q is processing queries per time unit. It is assumed that the number of reading operations on the tuple space $\Im_n$ is proportional to the number of tuples n. Hence,

$$S_M(q) := |\mathfrak{I}_n(q)| \text{ and } S_Q(q) := n \cdot \sum_{\substack{T\in\mathfrak{I} \\ q\in\delta_r(T)}} \pi_r(T) \ .$$

Let A_w, A_r and A_R denote the *average number of servers* taken into account while proceeding a writing, reading and bulk reading operation [8].

$$A_w := \sum_{T\in\mathfrak{I}} \pi_w(T)\cdot|\delta_w(T)| \text{ and } A_R := \sum_{T\in\mathfrak{I}} \pi_r(T)\cdot|\delta_r(T)| \ .$$

In bulk reading operations, every server in $\delta_r(T)$ has to be queried, even if a matching tuple already has been found on one server. However, a reading operation should stop after having found a tuple. Let $\chi: \{1,...,p\}\times\mathfrak{I} \to \{0,1\}$ denote the characteristic function which determines whether a given server holds a tuple that is matched by a given template. Furthermore, $\chi(T)$ is defined as the number of servers in $\delta_r(T)$ that hold a tuple which is matched by a given template. Then,

$$\chi(q,T) = 1 \ :\Leftrightarrow\ \exists T'\in\mathfrak{I}_n(q): \text{match}_{\mathfrak{I}}(T,T')$$
$$\chi(T) := \sum_{q\in\delta_r(T)} \chi_r(q,T) \ .$$

As a result, the expected number of servers queried is about $|\delta_r(T)|*[\max(1,\chi(T))]^{-1}$ for a template T. Hence,

$$A_r := \sum_{T\in\mathfrak{I}} \pi_r(T)\cdot\frac{|\delta_r(T)|}{\max(1,\chi(T))} \ .$$

Note, that the definition of S_Q is pessimistic, since it assumes that every query is a bulk reading operation. This is due to the fact that the ratio of reading and bulk reading operations is not defined in this model to simplify matters.

4.3 Conclusions of the Modeling

Server resources are limited, so that they scale up only to a certain degree. However, scalability means that the tuple space scales up, even for very large *n*. Therefore, the load of a server has to be independent of *n*.

Definition. The properties $S_M(q)$ and $S_Q(q)$ of the server *q* scale if and only if they are elements of O(1).

In analogy, *response times* should be independent of *n*.

Definition. The properties A_w, A_r and A_R scale if and only if they are elements of O(1).

Example. Whatever scheme is used, a scaling property opposes the scaling of another. E.g. in case of (δ^*,δ_1) used as distribution, $S_Q(q)$, A_r and A_R scale, but $S_M(q)$ and A_w do not. For (δ_1,δ^*) $S_M(q)$ and A_w scale, but $S_Q(q)$, A_r and A_R do not. If tuples are not distributed at all (p=1), then A_w, A_r and A_R scale, but $S_M(q)$ and $S_Q(q)$ do not.

Finally, a tuple space is called scalable, if all of its properties scale.

5. AN ADVANCED CONCEPT FOR SCALABILITY

As already mentioned before, one strictly relies on the systematic exploitation of structural restrictions, in order to conceive a scalable tuple space. More precisely, if the structure of the graph $(\mathfrak{I},\text{match}_{\mathfrak{I}})$ is known, similar tuples should be stored on the same server. Then, queries may be directed to servers that hold tuples similar to the template. However, such an approach requires a notion of similarity. E.g. hash functions can be used for this purpose [1], [10].

The structure of $(\mathfrak{I},\text{match}_{\mathfrak{I}})$ is implied by the matching on tuples. Therefore, an arbitrary $\text{match}_{\mathfrak{I}}$ hinders a systematic exploitation. In such a case, matching on fields is irrelevant and information about the structure of (F,match_F) cannot be used. Hence, the concept of this chapter assumes tuple space schemes in Ψ^*. Then, a formal or actual tuple is a vector of fields and matching on it is induced by matching on its fields. Therefore, similarity of tuples can be expressed as similarity of their fields.

This chapter introduces a new concept for scalability that fully exploits the structure of tuples. It consists of two steps. First, the structure of fields is taken into account by transforming them into a representation that is similar to hash codes. Although this transformation has to be implemented in addition, it is quite straightforward. In a second step, the structure of tuples is automatically deduced by the transformation to hypercubes. They are able to express similarity of tuples.

5.1 Intervals

The distribution based on hash functions is too coarse; it either maps to {q} or to {1,...,p}. The most general distribution maps to an arbitrary subset of {1,...,p}, but it takes O(p) for computation and storage. Therefore, a distribution has to map to manageable subsets of {1,...,p} that on the other hand have a sufficient fine granularity. It seems promising to use intervals for this purpose, because they may be represented in O(1) and are quite fine granular.

Let J(S) denote the set of intervals on an arbitrary total ordering S and $<_J$ a partial order on J(S) with

$$\forall U,V\in J(S): U <_J V \ \leftrightarrow\ \forall u\in U: \forall v\in V: u < v \ .$$

Assume that $\iota_{\mathfrak{I}}: \mathfrak{I} \to J(Nat)$ maps a tuple to an interval of natural numbers. In addition, $\iota_{\mathfrak{I}}$ has to comply with

$$\forall T_1,T_2\in\mathfrak{I}: \text{match}_{\mathfrak{I}}(T_1,T_2) \ \rightarrow\ \iota_{\mathfrak{I}}(T_1)\cap\iota_{\mathfrak{I}}(T_2) \neq \emptyset \ .$$

Furthermore, assume that $\iota_{\mathfrak{I}}$ complies with the inversion, that is

$$\forall T_1,T_2\in\mathfrak{I}: \iota_{\mathfrak{I}}(T_1)\cap\iota_{\mathfrak{I}}(T_2) \neq \emptyset \ \rightarrow$$
$$\text{match}_{\mathfrak{I}}(T_1,T_2) \vee \text{match}_{\mathfrak{I}}(T_2,T_1) \ .$$

Assume that there was such a $\iota_{\mathfrak{I}}$ in Figure 3*(a)*. Then,

$$\iota_{\mathfrak{I}}(T_j) \cap \iota_{\mathfrak{I}}(T_k) = \emptyset = \iota_{\mathfrak{I}}(S_j) \cap \iota_{\mathfrak{I}}(S_k) \ \text{ with } j,k\in\{1,2,3,4\} \text{ and } j\neq k \ .$$

If $\iota_{\mathfrak{I}}(T_1) < \iota_{\mathfrak{I}}(T_2) < \iota_{\mathfrak{I}}(T_3) < \iota_{\mathfrak{I}}(T_4)$, then $\iota_{\mathfrak{I}}(S_1) < \iota_{\mathfrak{I}}(S_2) < \iota_{\mathfrak{I}}(S_3)$, too. Therefore, $\iota_{\mathfrak{I}}(T_1) < \iota_{\mathfrak{I}}(S_2) < \iota_{\mathfrak{I}}(T_4)$, so that there is no valid value for $\iota_{\mathfrak{I}}(S_4)$, because $\iota_{\mathfrak{I}}(S_2) \subset \iota_{\mathfrak{I}}(S_4)$.

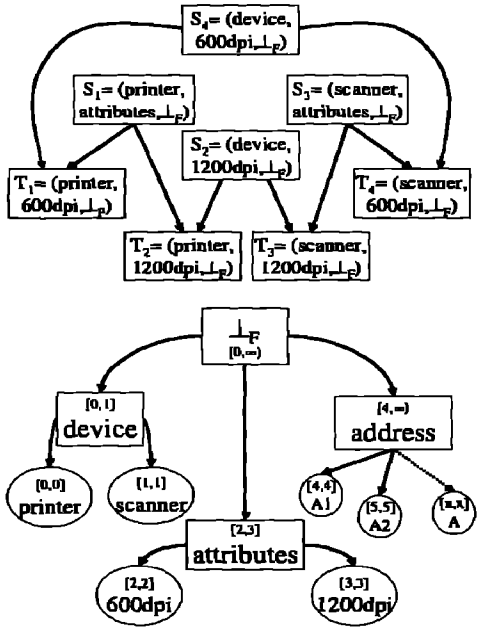

Figure 3. Excerpts of the graph $(\mathfrak{I},\text{match}_\mathfrak{I})$ above *(a)* and (F,match_F) below *(b)* in a service brokering scenario. Note, that the definition of a mapping to intervals is trivial, if the graph is a tree as in *(b)*.

However, Figure 3*(b)* suggests that it is no problem to map fields to intervals. This is due to the tree structure of (F,match_F). Furthermore, $\iota_\mathfrak{I}$ complies with

$$\forall f_1,f_2\in F:\ \text{match}_F(f_1,f_2) \leftrightarrow \iota_F(f_2)\subseteq\iota_F(f_1)\ .$$

In conclusion, the structure of tuples is too complex to be described by intervals. However, intervals may be used on fields.

5.2 Transformation of Tuples to Hypercubes

Let I_F denote the set of mappings $\iota_F: F\to J(Nat)$ that comply with

$$\forall f_1,f_2\in F:\ \text{match}_F(f_1,f_2)\ \to\ \iota_F(f_1)\cap\iota_F(f_2)\neq\varnothing\ .$$

Note, that I_F is not empty. Furthermore, let $I_F^C\subseteq I_F$ denote a subset of mappings $\iota_F\in I_F$ that in addition comply with

$$\forall f_1,f_2\in F:\ \text{match}_F(f_1,f_2)\ \to\ \iota_F(f_2)\subseteq\iota_F(f_1)\ ,$$
$$\forall f_1,f_2\in F:\ \iota_F(f_2)\subset\iota_F(f_1)\ \to\ \text{match}_F(f_1,f_2)\ ,$$

$$\forall f_1,f_2\in F:\ \iota_F(f_1)=\iota_F(f_2)\wedge f_1\neq f_2\ \to\ |\iota_F(f_1)|=1=|\iota_F(\inf(f_1,f_2))|$$

The last line is necessary, because a field f with $|\iota_F(f)|=1$ may match a set of other fields. If (F,match_F) is a tree, I_F^C is not empty.

The mapping of fields to intervals induces a mapping of tuples to hypercubes, as denoted by $I_\mathfrak{I}: I_F\to[\mathfrak{I}(F)\to J(Nat\cup\{-1\})^d]$. For an arbitrary $\iota_F\in I_F$ the mapping $\iota_\mathfrak{I}:=I_\mathfrak{I}(\iota_F)$ is induced with

$$\forall t\in\tau(F):\ \iota_\mathfrak{I}(t)=\iota_F(\Pi_1(t))\times\ldots\times\iota_F(\Pi_{|t|}(t))\times[-1,-1]^{d-|t|}\ ,$$

$$\iota_\mathfrak{I}(\bot_\mathfrak{I})\ =\ [0,\infty)\times[-1,\infty)^{d-1}\ .$$

Therefore, tuples are mapped to hypercubes with d dimensions. E.g. for the mapping ι_F of Figure 3*(b)*, it is $I_\mathfrak{I}(\iota_F)(\bot_\mathfrak{I})=[0,\infty)\times[-1,\infty)^2$ and

$$I_\mathfrak{I}(\iota_F)\ (\ (\text{printer,attributes,address})\ )=[0,0]\times[2,3]\times[4,\infty)\ .$$

Theorem 5.2.1. For $(F,\text{match}_F,\mathfrak{I},\text{match}_\mathfrak{I})\in\Psi^*$ and $\iota_\mathfrak{I}=I_\mathfrak{I}(\iota_F)$ with $\iota_F\in I_F$, it is

a) $\forall t_1,t_2\in\tau:\ \text{match}_\mathfrak{I}(t_1,t_2)\to\iota_\mathfrak{I}(t_1)\cap\iota_\mathfrak{I}(t_2)\neq\varnothing$

b) $\forall t_1,t_2\in\mathfrak{I}:\ \text{match}_\mathfrak{I}(t_1,t_2)\to\iota_\mathfrak{I}(t_1)\cap\iota_\mathfrak{I}(t_2)\neq\varnothing$.

Proof.

a) $|t_1|=n=|t_2|$ and it is $\text{match}_F(\Pi_j(t_1),\Pi_j(t_2))$ for an arbitrary j with $1\le j\le n$. Therefore, $\iota_F(\Pi_j(t_1))\cap\iota_F(\Pi_j(t_2))\neq\varnothing$ and it follows $\iota_\mathfrak{I}(t_1)\cap\iota_\mathfrak{I}(t_2)\neq\varnothing$.

b) Because of $\forall t\in\tau:\ \iota_\mathfrak{I}(t)\subseteq\iota_\mathfrak{I}(\bot_\mathfrak{I})$ the direct outcome of a).

Compared to I_F, I_F^C does not enhance correlation of hypercubes to the matching of tuples. E.g. for $\iota_\mathfrak{I}(t_1)=[0,2]\times[0,0]$, $\iota_\mathfrak{I}(t_1)=[0,2]\times[1,2]$ and $\iota_\mathfrak{I}(t_3)=[1,1]\times[0,1]$, it is $\iota_\mathfrak{I}(T)=[0,2]\times[0,2]$ with $T=\{t_1,t_2\}$. Then, $\Pi_j(\iota_\mathfrak{I}(t_3))\subset\Pi_j(\iota_\mathfrak{I}(T))$ for $j\in\{1,2\}$, but $\text{match}_\mathfrak{I}(T,t_3)$ is false.

5.3 Distribution Based on Hypercubes

The transformation of tuples to hypercubes abstracts from tuples, however without ignoring the structure of tuples that is induced by matching. Hence, the tuples may be distributed based on their hypercubes, which gives more room for differing distribution strategies. This section suggests two of them. The first one is to map a hypercube to a set of natural numbers that are interpreted as hash codes. The other strategy introduces adaptivity into the distribution, since it takes into consideration, which tuples are stored in the tuple space. Therefore, every server is assigned a hypercube that identifies its *tuple domain*. Then, the distribution is adaptive, e.g. by splitting domains that are frequently used. In the following, $\iota_F\in I_F$ is assumed.

Hash Codes. Let $G: J(Nat\cup\{-1\})^d\to P(Nat)$ denote a mapping of a hypercube to a set of hash codes. E.g. such a mapping can be determined with Gödel numbering, that is

$$G(S):=\{\ \prod_{j=1}^{d}p_j^{1+s_j}\ |\ (s_1,\ldots,s_d)\in S\ \}$$

with $\{p_1,p_2,\ldots\}$ depicting the set of prime numbers. Then, the assignment $\delta_G\in\Delta$ of a tuple to a set of servers is performed by an arbitrary hash function. E.g. based on [6], it is

$$\delta_G(T):=\{\ 1+[x\bmod p]\ |\ x\in G(I_\mathfrak{I}(\iota_F)(T))\ \}\ .$$

Theorem 5.3.1. δ_G is a permissible distribution.

Proof. If $\text{match}_\mathfrak{I}(T_1,T_2)$, Theorem 5.2.1(b) shows $I_\mathfrak{I}(\iota_F)(T_1)\cap I_\mathfrak{I}(\iota_F)(T_2)\neq\varnothing$. Then, there is an $x\in G(I_\mathfrak{I}(\iota_F)(T_1))\cap G(I_\mathfrak{I}(\iota_F)(T_2))$. Hence, $\delta_G(T_1)\cap\delta_G(T_2)\neq\varnothing$.

Figure 4 illustrates this concept. Note, that it is the generalization of the distribution based on hash functions, since they are identical in case of

$$\forall T\in\mathfrak{I}:\ [\ |I_\mathfrak{I}(\iota_F)(T)|=1\ \vee\ I_\mathfrak{I}(\iota_F)(T)=I_\mathfrak{I}(\iota_F)(\bot_\mathfrak{I})\ ]\ .$$

However, this distribution strategy has to be refined, because $\delta_G(T)$ takes $O(|I_\mathfrak{I}(\iota_F)(T)|)$ in computation complexity and, for an arbitrary mapping G, $|\delta_G(T)|$ takes $O(|I_\mathfrak{I}(\iota_F)(T)|)$, too. Furthermore, the servers' tuple domains do not adapt to the usage

profile. For many mappings G, it is costly to adjust the number of servers.

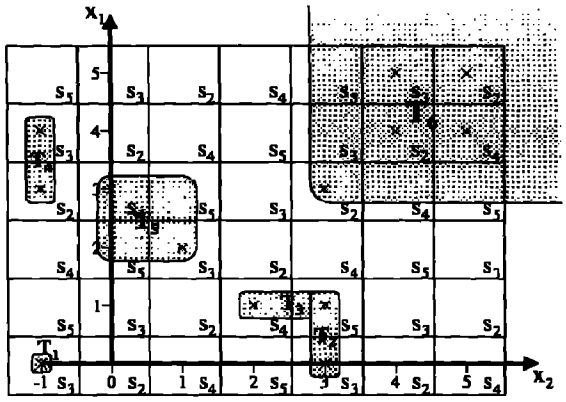

Figure 4. Distribution strategy based on hashing hypercubes for five servers $\{s_1,...,s_5\}$. The example shows tuples with one or two dimensions that are mapped to rectangles as induced by ι_F of Figure *3(b)*. The displayed tuples are T_1=(printer), T_2=(device,1200dpi), T_3=(scanner,attributes), T_4={(1200dpi),(A1)}, T_5={(1200dpi,printer),(600dpi,scanner)} and T_6={(address,address),(1200dpi,1200dpi)}. δ_G is based on Gödel numbering, so that $\delta_G(T_1)=\{3\}$, $\delta_G(T_2)=\{3,5\}$, $\delta_G(T_3)=\{4,5\}$, $\delta_G(T_4)=\{3,4,5\}$, $\delta_G(T_5)=\{2,3\}$ and $\delta_G(T_6)=\{2,3,4,5\}$.

Tuple Domains. For each server with the index q, Σ_q denotes its hypercube. The servers' hypercubes have to comply with

$$\bigcup_{q=1}^{p}\Sigma_q = [0,\infty)\times[-1,\infty)^{d-1} \wedge \forall q,q'\in\{1,...,p\}: \Sigma_q\cap\Sigma_{q'} = \emptyset .$$

Then, tuples T with $I_\mathfrak{I}(\iota_F)(T)\cap\Sigma_q$ identify the tuple domain of server q. Hence, let $\delta_\Sigma\in\Delta$ be defined as

$$\delta_\Sigma(T) := \{\, q\in\{1,...,p\} \mid I_\mathfrak{I}(\iota_F)(T)\cap\Sigma_q\neq\emptyset \,\} .$$

Theorem 5.3.2. δ_Σ is a permissible distribution.

Proof. If $\text{match}_\mathfrak{I}(T_1,T_2)$, Theorem 5.2.1(b) shows that there is a $x\in I_\mathfrak{I}(\iota_F)(T_1)\cap I_\mathfrak{I}(\iota_F)(T_2)$. Therefore, $I_\mathfrak{I}(\iota_F)(T_1)\cap I_\mathfrak{I}(\iota_F)(T_2)\subseteq[0,\infty)\times[-1,\infty)^{d-1}$ implies that there is a q with $x\in\Sigma_q$. Hence, $q\in\delta_\Sigma(T_1)\cap\delta_\Sigma(T_2)$.

This distribution strategy is illustrated in Figure 5. Note, that $I_\mathfrak{I}(\iota_F)(T)\cap\Sigma_q=\emptyset$ implies that there is no tuple stored on server q that is matched by template T.

Unlike the other suggested strategies, the servers' state is taken into account. Therefore, it is possible to automatically adapt the distribution to the usage profile of the tuple space: If the number of tuples that are stored on server q exceeds $\max_\Sigma$, the tuple domain of q is split and one additional server is added. If there are only few tuples stored on two servers with adjacent tuple domains, the domains are merged.

However, there are some problems when implementing this strategy. The program units that compute $\delta_\Sigma(T)$ need to know about the servers' tuple domains. Furthermore, the computation has to verify for every q, whether the intersection of $I_\mathfrak{I}(\iota_F)(T)$ and Σ_q is empty. If *priority search trees* [5] are used, $\delta_\Sigma(T)$ takes $O((\log p)^{d-2})$ in computation complexity.

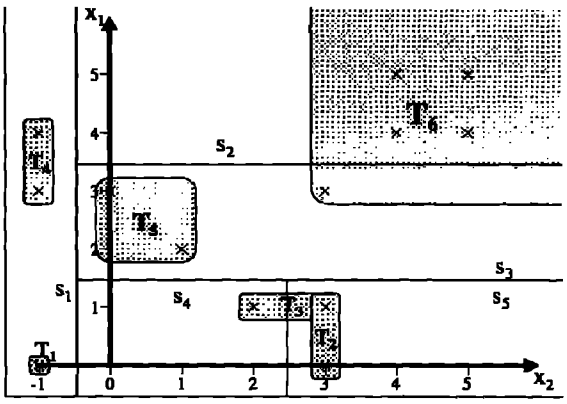

Figure 5. Distribution strategies based on tuple domains for five servers $\{s_1,...,s_5\}$. The tuples and ι_F are the same as in Figure 4. However, the distribution strategy based on tuple domains is applied. The servers' rectangles are $\Sigma_1=I_\mathfrak{I}(\iota_F)((\perp_F))$, $\Sigma_2=I_\mathfrak{I}(\iota_F)((\text{address},\perp_F))$, $\Sigma_3=I_\mathfrak{I}(\iota_F)((\text{attributes},\perp_F))$, $\Sigma_4=I_\mathfrak{I}(\iota_F)(\{(\text{device},\text{device}),(\text{device},\text{600dpi})\})$ and $\Sigma_5=I_\mathfrak{I}(\iota_F)(\{(\text{device},\text{1200dpi}),(\text{device},\text{address})\})$. Therefore, the tuples are distributed to $\delta_\Sigma(T_1)=\{1\}$, $\delta_\Sigma(T_2)=\{5\}$, $\delta_\Sigma(T_3)=\{4,5\}$, $\delta_\Sigma(T_4)=\{1\}$, $\delta_\Sigma(T_5)=\{3\}$ and $\delta_\Sigma(T_6)=\{2,3\}$.

5.4 Analysis of the Concept

The quality of the distribution strategies highly depend on ι_F. E.g. for a constant ι_F, no property of the tuple space scales. However, it is impossible to define ι_F automatically, if match_F is user defined. Even though the application programmer has to implement ι_F, it is by far an easier task compared to the implementation of a hash function. Furthermore, if (F,match_F) is structured as a tree, it should be feasible to define a $\iota_F\in I_F^C$ automatically, as shown in Figure *3(b)*. Then, it is a sufficient condition for a scaling A_w, that only actual tuples are stored in the tuple space. This is true for many application areas and, besides, it is a basic assumption of the approaches of section 2.2.

The finest granularity in the hypercube concept are points. All tuples that share one point are stored on the same server. This has to be taken into account in the definition of ι_F.

Theorem 5.4.1. Let δ_Σ or δ_G be the distribution and $\iota_F\in I_F$. If there exists an $x\in[0,\infty)\times[-1,\infty)^{d-1}$ with

$$\exists\mathfrak{I}'\subseteq\mathfrak{I}_n: |\mathfrak{I}'|\in\omega(1)\wedge\forall T\in\mathfrak{I}': x\in I_\mathfrak{I}(\iota_F)(T) ,$$

then $S_M(q)$ does not scale for a server q.

Proof. If δ_Σ is applied, let q denote the server with $x\in\Sigma_q$. If δ_G is applied, let q denote the server as induced by $G(\{x\})$. Then, it is $\mathfrak{I}'\subseteq\mathfrak{I}_n(q)$, hence $|\mathfrak{I}_n(q)|\in\omega(1)$.

The effectiveness of the distribution strategy based on hashing hypercubes has still to be examined. It strictly depends on an appropriate mapping G, especially in regard to the dynamic

behaviour of the tuple space. Hence, the rest of this section is focused on the distribution δ_Σ that is based on tuple domains.

Theorem 5.4.2. Let δ_Σ be the distribution based on tuple domains and $\iota_F \in I_F$. If there is no $x \in [0,\infty) \times [-1,\infty)^{d-1}$ with

$$\exists \mathfrak{I}' \subseteq \mathfrak{I}_n: |\mathfrak{I}'| \in \omega(1) \wedge \forall T \in \mathfrak{I}': x \in I_{\mathfrak{I}}(\iota_F)(T),$$

then $S_M(q)$ scales for every server and adjusting the number of servers takes O(1).

Proof. Assume that $S_M(q) \in \omega(1)$ for a server q. Then, its tuple domain is a single point $\{x\}$, otherwise it would have been split. Hence, $|\mathfrak{I}_n(q)| = S_M(q) \in \omega(1)$ and $\forall T \in \mathfrak{I}_n(q): x \in I_{\mathfrak{I}}(\iota_F)(T)$. If tuple domains are merged or split, only two servers are concerned, so that adjustment is done in O(1).

An analysis of A_r requires an explicit definition of ι_F and of the algorithm that merges and splits tuple domains. However, such an algorithm has still to be researched in the future. If the analysis of A_r proves to be too complex, simulative methods may be applied.

5.5 Conclusions and Future Work

The paper has presented a formal description of tuple spaces which takes into account different levels of expressiveness in current tuple space implementations. Distribution strategies are formally characterized and a deterministic model of scalability is introduced.

The formalization provides the foundation of a new concept for rendering tuple spaces scalable. The concept introduces an adaptive distribution of tuples based on an intermediate representation, i.e. hypercubes. Furthermore, it generalizes former approaches towards scalability and thus it overcomes some of their limitations.

Future work is in direction of implementing a tuple space based on the hypercube concept, in order to verify its feasibility and effectiveness.

6. REFERENCES

[1] Bjornson, R. D.: Linda on Distributed Memory Multiprocessors. PhD thesis, Yale University, TR931 (1993)

[2] Castellani, S, Ciancarini, P., Rossi, D.: The ShaPE of ShaDE: a Coordination System. Technical Report UBLCS, Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy (1995)

[3] Cohn, P., M.: Algebra, John Wiley & Sons, Second Edition (1982)

[4] Corradi, A., Leonardi, L., Zambonelli, F.: A Scalable Tuple Space Model for Structured Parallel Programming. Proceedings of the Conference on Massively Parallel Programming Models, IEEE CS Press, Pages 25-32, Berlin, Germany (1995)

[5] McCreight, E. M.: Priority Search Trees, SIAM J. Computing 14, Pages 257-276 (1985)

[6] ECOM (Ed.): Electronic Commerce – An Introduction. http://ecom.fov.uni-mb.si (1998)

[7] Gaedke, M., Turowski, K.: Generic Web-Based Federation of Business Application Systems for E-Commerce Applications. In: S. Conrad; W. Hasselbring; G. Saake (Ed.): 2nd Intl. Workshop on Engineering Federated Information Systems (EFIS99), Germany (1999)

[8] Gelernter, D.: Generative communication in Linda. ACM Transactions on Programming Languages and Systems, 7(1): 80-112 (1985)

[9] Gelernter, D.: Multiple tuple spaces in Linda. In E. Odijk, M. Rem, and J.-C. Syre, editors, PARLE '89: Parallel Architectures and Languages Europe. Volume II: Parallel Languages, volume 366 of Lecture Notes in Computer Science, pages 20-27. Springer-Verlag (1989)

[10] Larsen, J. E., Spring, J. H.: A Dynamically Fault-Tolerant and Dynamically Scalable Distributed Tuplespace for Heterogeneous, Loosely Coupled Networks (GLOBE), Master thesis, University of Copenhagen (1999)

[11] Luger, et al.: The Blackboard Architecture for Problem Solving, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Second Edition, Chapter 5.5, Benjamin/Cummings Publishing Company (1993)

[12] Rowstron, A.: WCL, a Coordination Language for Geographically Distributed Agents. World Wide Web Journal, Volume 1, Issue 3, 167-179 (1998)

[13] Sun Microsystems: JavaSpaces Technology, http://java.sun.com/products/javaspaces (2000)

[14] Sun Microsystems: Java, http://java.sun.com (2000)

[15] Weiser, M.: Some Computer Science Issues in Ubiquitous Computing, Communications of the ACM (1993)

[16] Wells, G., Chalmers, A.: An Extended Linda System Using PVM. In PVM Users' Group Meeting, Pittsburgh (1995)

[17] Wyckoff, P., McLaughry, S. W., Lehman, T. J., Ford, D. A.: TSpaces, IBM Systems Journal (1998)