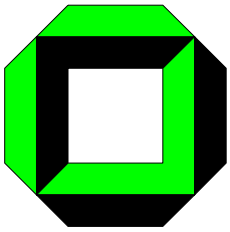
Analysis and Conception of Tuple Spaces in the Eye of Scalability

Philipp Obreiter

obreiter@ipd.uni-karlsruhe.de

November 4, 2003

Technical Report Nr. 2003-20



University of Karlsruhe

Faculty of Informatics

Institute for Program Structures and Data Organization

D-76128 Karlsruhe, Germany

Contents

1 Introduction

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

This report is an English version of the author's diploma thesis. It comprises the chapter two, three, four, and five. By this means, the design and the implementation of the proposed tuple space is not part of this report.

2 State of the Art in Tuple Spaces

Tuple spaces have been conceived and extended under the influence of databases and messaging systems. However, research has not resolved yet how to render tuple spaces scalable.

2.1 The Tuple Space Concept and its Extensions

A *tuple space* [32] is a logically shared associative memory that enables cooperation based on the blackboard design pattern [43]. *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, 14, 31, 33, 60]. E.g. *object orientation* has been introduced to tuple spaces [14] and [31] suggests the use of *semantic* templates that match tuples structurally. In Bauhaus Linda [13], it is possible to define *nested* tuples. Hence, tuples are not structured as collections any more, but they are trees with fields as their leaves. There are several implementations of tuple spaces, e.g. Linda [32], JavaSpaces [57] and T Spaces [61]. They differ in the amount of extensions implemented. A more detailed introduction to tuple spaces is given in [46].

2.2 Related Research Areas

The interaction model of tuple spaces uncouples its participants. Hence, it is possible to gap their time and space remoteness. There are concepts in other research areas that aim at the same goal. However, they differ in their data and interaction models.

Messaging systems like JMS [38] pass messages through information channels. The participating entities have to subscribe to information channels prior to receiving messages. Hence, temporal uncoupling is not provided in general. Furthermore, there is no concept of data retrieval, but the information channel determines the nature of the data that is received. On the other hand, it has been widely acknowledged that messaging systems are inherently scalable. This ensues from the restriction that information channels have to be defined before the submittance of a message. In general, the set of information channels can be partitioned into non overlapping subsets. Hence, parallelization and scalability may be achieved.

Historically, databases have been invented, in order to gap time remoteness. Nowadays, remote access to databases has become fairly standard, so that databases come near to tuple spaces. On the other hand, tuple spaces virtually have

become a database due to recent extensions of its model. However, data retrieval in databases is based on the evaluation of queries. Various query languages like SQL [22] have been developed. In principle, they are not bound to the underlying data model. In addition, query languages are more expressive than templates, e.g. queries may be nested and operations can be performed on the result set. In contrast, the original concept of data retrieval in tuple spaces assumes that only one tuple, if any, is returned. Furthermore, template matching lacks the arbitrary use of predicates, as it is common in query languages.

As for the data model, relational databases [18] expect that every tuple complies with an arbitrary but predefined scheme. The a priori knowledge of key values in relational data schemata enables the application of indexing algorithms, so that data retrieval is more efficient than in tuple spaces. There are several suggestions that extend or replace the relational data model. They are reflected in recent extensions of the tuple space model. Apart from suggesting its own object oriented data model [41], object orientation has lead to an extension of the relational data model, i.e. the object relational data model [56]. In object oriented data bases, an identity may be assigned to data, and the hierarchy of the type system is extended to class hierarchies. Furthermore, object orientation introduces monomorphy to operators, as it is reflected by customizable matching in object oriented tuple spaces. The NF^2 model [52] is yet another proposal that extends the relational data model. The domain of tuples is not restricted to atomic values any more, the domain may be a relation. Hence, the data model is extended recursively. In contrast to nested tuples in Bauhaus Linda, the NF^2 model expects that the data scheme is predefined. Therefore, the definition of keys and indices in the relational model may be extended to the NF^2 model. Recently, this restriction has been relaxed by proposing semi structured data models like OEM [35] and XML [10]. Semi structured data schemes are specified by context-free grammars, if they are defined at all. Otherwise, the data scheme is implicitly given by the data. Therefore, semi structured data is irregular. E.g. tuple spaces may be regarded as semi structured database, if nested tuples are taken into account. As a result, the concept of keys and indexing is not directly applicable to semi structured databases. Furthermore, query languages like Lorel [1] support structural queries, since the structure contains information and varies from data to data. Hence, the expressiveness of data retrieval is enhanced, which is reflected by the introduction of semantic templates [31] to the tuple space model.

The interaction model of databases has been extended, in order to support the push model of message passing systems. The *push model* enables system components to be notified of specified events. It is supported in some tuple space implementations, e.g. JavaSpaces and TSpaces.

In conclusion, extensions of the tuple space model are derived from other research areas. In Figure 1, such dependencies are listed. Note, that the original tuple space data model corresponds with the relational data model.

Research Area	Influence on	Derived Concept
relational databases	data model	tuples
object-relational databases		class hierarchies
	data retrieval	customizable matching
		matching hierarchies
semi structured databases		semantic templates
NF^2 model	data model	nested tuples
messaging systems	interaction model	notification service

Figure 1. The tuple space model and its extensions have been conceived under the influence of other research areas. A list of such dependencies is shown.

2.3 Prior Studies of Scalability in Tuple Spaces

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 [50], 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 [20] 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 [8] 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. This problem has not been solved yet, as it is shown in chapter 4.

Alternatively, the tuple space run-time system dynamically adapts the indexing of tuples to the usage profile, as it is true for Paradise [9]. It keeps record whether the fields of a stored tuple are matched by formal or actual fields of a template. Then, the indexing is based on fields, that are not frequently matched by templates' formal fields. This strategy requires re-indexing, in order to adapt to time variant use of templates. If the tuples are stored on different servers, then the re-indexing requires replacement of the tuples among the servers. Hence, this approach does not fit well to a distributed tuple space that aims at scalability.

A completely different approach is presented by Safer Tuple Spaces [36] and by Enterprise TSpaces [39]. It imposes additional structural restrictions on the spaces. E.g. each space is only allowed to hold tuples of a predefined type, and it can be specified which fields of a template have to be actual fields. Hence, tuple spaces become similar to databases, so that performance gains may be achieved, e.g. by indexing. However, the popularity of tuple spaces is partly due to its flexibility that comes from its ad hoc structure and the lack of scheme definitions. Therefore, the presented restrictions seem to be too severe. Yet it seems to be a promising idea to include additional information from the tuple spaces' environment.

3 Analysis of Tuple Spaces

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 [32] has recently been extended by object orientation, semantic tuples [31] and nested tuples [13]. As a result, tuple spaces are more expressive, but important structural restrictions are set aside. E.g. in JavaSpaces [57] and TSpaces [61] 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. Afterwards, the formalization given in this chapter is compared to other approaches.

3.1 Formalization of Tuple Spaces

In a first step, fields and tuples are formalized in a way, that integrates extensions. One key concept is to regard templates as tuples [13, 31], 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 trees, with fields as their leaves. 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-inheritance 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 [58] $\perp_F$ 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 $\text{class}: F \to C$ denote the mapping $\text{class}(c) = c = \text{class}(i)$ 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 fields. E.g. in [57, 61] 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. Furthermore, let $\tau(F)$ denote the set of formal and actual tuples. If nested tuples are allowed, then $\tau(F)$ and $\tau_{actual}(F)$ are inductively defined by

$$(x_1,...,x_n) \in (\tau_{actual}(F) \cup I)^n \Rightarrow (x_1,...,x_n) \in \tau_{actual}(F) ,$$
$$(x_1,...,x_n) \in (\tau(F) \cup F)^n \Rightarrow (x_1,...,x_n) \in \tau(F) .$$

If tuples are vectors of fields, then

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

with $d \in Nat \cup \{\infty\}$ depicting the maximal dimension of tuples. Note, that this is a subset of the set of nested tuples. In both cases, the set of formal tuples is defined as $\tau_{formal}(F) = \tau(F) \setminus \tau_{actual}(F)$. Furthermore, the *projection* $\Pi_i: \tau(F) \rightarrow \tau(F) \cup F$ is defined by

$$(x_1,...,x_n) \in \tau(F) \Rightarrow \Pi_i((x_1,...,x_n)) := x_i .$$

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

$$\perp_{\Im(F)} = \tau(F) \text{ and } \perp_{\Im(F)} \in \Gamma(F) \subseteq P(\tau(F)) \setminus \{\varnothing\}$$

with P(A) depicting the power set of A. Note, that $\Gamma(F)$ is not completely defined by the condition, in order to reflect different support of semantic tuples in current implementations. Finally, $\Im(F) := \Gamma(F) \cup \tau(F)$ 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, $\Im_P(F)$ is a superset of $\Im(F)$, as it is defined as $\tau(F) \cup P(\tau(F)) \setminus \{\varnothing\}$. In addition, let $\gamma: \Im(F) \rightarrow P(\tau(F))$ denote the mapping that is the identity mapping on $\Gamma(F)$, and for $t \in \tau(F)$ it is defined as $\gamma(t) := \{t\}$. It is extended to $\gamma_P: P(\Im(F)) \rightarrow P(\tau(F))$ with

$$\forall \Im' \subseteq \Im(F): \gamma_P(\Im') = \bigcup_{T \in \Im'} \gamma(T) .$$

Matching. Let $match_\Im$ denote the *matching relation* on tuples. In order to be as expressive as in [57, 61], no restriction for the matching on tuples is applied, except of $match_\Im \subseteq \Im(F)^2$. However, the original concept [32] of matching is to match the fields of a template to the one of a tuple. This is expressed by $\omega_\tau \subseteq \tau(F)^2$ that is defined inductively by

$$(r,s)\in\omega_\tau \;\Leftrightarrow\; r=\perp_F \vee [\, r=(r_1,...,r_n) \wedge s=(s_1,...,s_n) \wedge \forall i\in\{1,...,n\}: (s_i\in F \wedge match_F(r_i,s_i)\,) \vee (r_i,s_i)\in\omega_\tau \,] .$$

Then, the definition of ω_τ is extended to sets of actual and formal tuples by

$$\omega_P := \{ (T_1,T_2)\in\Im_P(F)^2 \mid \forall t_2\in\gamma(T_2): \exists t_1\in\gamma(T_1): \omega_\tau(t_1,t_2) \} .$$

In conclusion, $\omega := \omega_P \cap \Im(F)^2$ is the appropriate definition on tuples. Note, that $\omega(\perp_{\Im(F)},x)$ is true for all $x\in\Im(F)$.

Signatures. $\sigma: \tau(F) \to \tau_{formal}(F)$ is defined inductively by

$$t=(x_1,...,x_n) \wedge n=|\sigma(t)| \wedge \forall i\in\{1,...,n\}: y=\Pi_i(\sigma(t)) \wedge [\, (\, x_i,y\in F \wedge class(x_i)=y\,) \vee \sigma(x_i)=y \,]$$

with $t\in\tau(F)$. $\sigma(t)$ is called the *signature* of $t\in\tau(F)$. A set of tuples with the same signature is denoted by

$$t^\sigma := \{\, t'\in\tau(F) \mid \sigma(t) = \sigma(t'\ )\,\} .$$

$\Im^\sigma(F)$ is the *quotient set* [17] of $\Im(F)$ as implied by

$$\tau^\sigma(F) := \{\, t^\sigma \mid t\in\tau(F)\,\} \quad\text{and}\quad \Gamma^\sigma(F) := \{\, t^\sigma \mid T\in\Gamma(F) \wedge t\in T\,\} .$$

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.

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 [17]. Semantic, formal and actual fields or tuples are represented as rhombi, rectangles and circles respectively. In the following, reflexivity and transitivity is omitted 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 2 shows an example of a scheme.

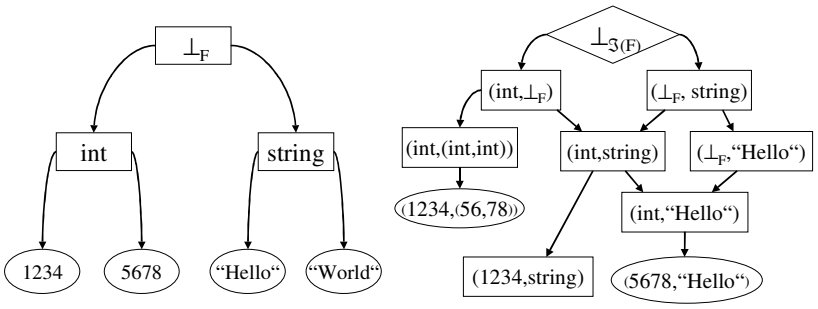


Figure 2. Excerpt of an exemplary nested tuple space scheme that can be used in Bauhaus Linda. $(F,match_F)$ is shown on the left and $(\Im(F),match_\Im)$ on the right.

Alternatively, tuples may be visualized based on the graph $(\tau(F),match_\Im)$. Then, semantic tuples are represented as hypergraphs [17].

3.2 A Taxonomy of Tuple Space Schemes

Tuple space scheme are very expressive. Except for multiple inheritance and semantic and nested tuples, JavaSpaces [57] and T Spaces [61] allow such schemes to be implemented. However, Linda [32] does not support object orientation, user defined field matching, semantic and nested tuples and user defined tuple matching. On the other hand, schemes that are allowed in Linda are by far better structured, e.g. $match_\Im=\omega$ is true for these schemes. Therefore, an analysis of tuple spaces should take into account different levels of restrictions on F, $match_F$, $\Gamma(F)$, $\tau(F)$ and $match_\Im$. As a result, there are five degrees of freedom.

This contribution introduces a *taxonomy* which is aware of different levels of restrictions. The degrees of freedom are labeled *A*, *B*, *C, D* and *E*. A tuple space scheme is characterized by five index numbers. A scheme with no additional restrictions imposed upon has the index zero on each axe. The more restrictive a scheme is, the higher is its index number of the according axe. E.g. a scheme with A=1 does not allow multiple inheritance. Ψ_{ABCDE} is defined as the set of the tuple space schemes with the indices A, B, C, D and E respectively, e.g. $\Psi_{00000}=\Psi$. Furthermore, Ψ_{AB} depicts the set of fields $(F,match_F)$ with appropriate indices on the A-axe and B-axe. In the following, the five degrees of freedom and the definition of their indices are introduced.

Class Hierarchy (the A-Axe). Initially, multiple inheritance is allowed. However, JavaSpaces [57] and T Spaces [61] expect an inheritance tree which prohibits multiple inheritance. Furthermore, Linda [32] is not object oriented, so that no inheritance is supported.

A=1: Multiple inheritance is not allowed. $(C,\leq_c)$ is a tree [17].

A=2: Apart of $\bot_F$, no inheritance is allowed, so that $C\setminus\{\bot_F\}$ is an anti-chain [17]. Furthermore, no instances are allowed for $\bot_F$. Note, that $\bot_F$ is the `void` type in some non object oriented programming languages.

Field Matching (the B-Axe). $match_F$ is already fully defined for formal fields. However, there is no restriction on how actual fields match other fields. E.g. actual fields could match $\bot_F$. Hence, matching on fields is not necessarily acyclic. Let $match_F^*$ denote the transitive hull [17] of $match_F$, that is

$$match_F^*(f_1,f_2) :\Leftrightarrow f_1= f_2 \vee \exists x_0,...,x_n\in F:$$
$$x_0=f_1 \wedge x_n=f_2 \wedge \forall i\in\{1,...,n\}: match_F(x_{i-1}, x_i) .$$

$match_F^*$ induces the *equivalence relation* ~ and the *quotient set* $F^{\sim}$ with

$$f_1\sim f_2 :\Leftrightarrow match_F^*(f_1,f_2) \wedge match_F^*(f_2,f_1) .$$

The relation $match_{F^\sim}\in F^{\sim 2}$ is well-defined by

$$\text{match}_{F^{\sim}}(f^{\sim}_1, f^{\sim}_2) :\Leftrightarrow \text{match}_F^{*}(f_1, f_2)$$

and $\leq_{F^{\sim}} := \text{match}_{F^{\sim}}$ is a *partial order* on $F^{\sim}$. Note, that $F^{\sim}$ is acyclic.

B=1: match_F is transitive, i.e. $\text{match}_F = \text{match}_F^{*}$. In addition, actual fields may only match actual fields of the same class, i.e. $\forall i \in I_c: \forall f \in C \setminus I_c: (i,f) \notin \text{match}_F$. Furthermore, $(\kappa^{\sim}, \text{match}_{\kappa^{\sim}})$ is structured as a tree for an arbitrary class $c \in C$ and $\kappa := \{c\} \cup I_c$. E.g. the scheme of Figure 3*(a)* is B=0.

B=2: In addition, ~ is the identity mapping on F, i.e. $f^{\sim} = \{f\}$ for an arbitrary field *f*. Then, $\leq_F := \text{match}_F$ is a partial order. E.g. the scheme of Figure 3*(b)* is B=1.

B=3: An actual field may not match any other actual field. Hence, $\leq_F \cap I^2$ is the identity mapping on I. Note, that matching based on equality [32] fulfills this restriction.

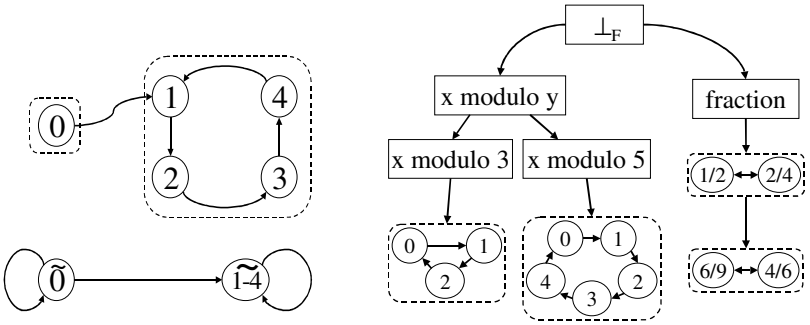


Figure 3. The left *(a)* gives an example of the transformation between F *(above)* and its quotient set $F^{\sim}$ *(below)*. On the right *(b)*, the significance of the B=1 restriction is shown. Therein, (F, match_F) only complies with B=1, but not with B=2. Matching is induced by the successor relationship in the *x modulo y* instances. For *fraction* instances, matching is induced by the smaller-or-equals relationship. Note, that only an excerpts of F and $F^{\sim}$ are shown in *(a)* and *(b)*.

Semantic Tuples (the C-Axe). The only semantic tuple [31] supported in current tuple space implementations [57, 61, 32] is $\perp_{\mathfrak{I}(F)}$. Semantic tuples are very powerful, since the power set $P(\tau(F))$ is uncountable [17].

C=1: The only semantic tuple is $\perp_{\mathfrak{I}(F)}$.

Nested Tuples (the D-Axe). Most tuple space implementations [57, 61, 32] do not directly support nested tuples.

D=1: A tuple is vector of fields and there exists a maximal dimension *d*, i.e.

$$\forall t \in \tau(F): [\ |t| \leq d\ \wedge \forall i \in \{1,...,|t|\}: \Pi_i(t) \in F\ ] .$$

Tuple Matching (the E-Axe). According to the definitions of the previous section, tuple matching can be arbitrarily defined.

E=1: Matching on tuples is defined by ω, i.e. $match_\Im = \omega$.

Examples. Sets of schemes are characterized as in Figure 4. The larger the area of the polygon, the more expressive and the less structured the schemes are. Current tuple space implementations extend the expressiveness of tuple spaces in different directions. However, Linda schemes remain the smallest common divisor, as it is illustrated in the figure.

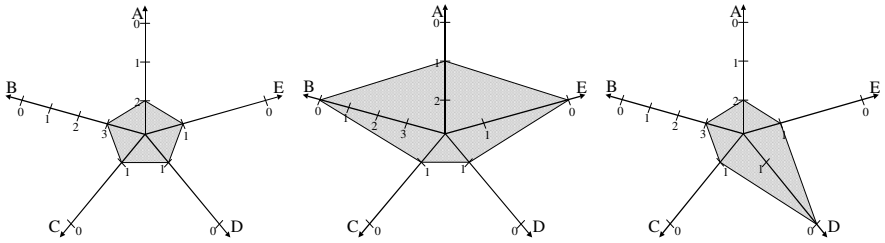


Figure 4. The taxonomy classifies Linda schemes as Ψ_{23111} *(left)*, JavaSpaces and T Spaces schemes as Ψ_{10110} *(middle)* and Bauhaus schemes as Ψ_{23101} *(right)*. JavaSpaces and T Spaces schemes support object orientation without multiple inheritance (A=1). In addition, matching is customizable (B=E=0). Bauhaus Linda introduces nested tuples to Linda (D=0).

Finally, it is shown that the taxonomy captures the level of structural restrictions.

Theorem 3.2.1. $\Psi_{ABCDE} \subseteq \Psi_{A'B'C'D'E'} \Leftrightarrow [\, A' \le A \wedge B' \le B \wedge C' \le C \wedge D' \le D \wedge E' \le E \,]$.

Proof. Let $\psi \in \Psi_{ABCDE}$ and $A' \le A$, $B' \le B$, $C' \le C$, $D' \le D$ and $E' \le E$. Then, ψ complies with the lesser strict conditions of *A′ B′ C′ D′* as *E* already shown for each axe. Hence, $\psi \in \Psi_{A'B'C'D'E'}$. Let $A' > A$, $B' > B$, $C' > C$, $D' > D$ or $E' > E$. As it is clear from the taxonomy's definition, the conditions on each axe express distinct levels of strictness. Hence, there has to be tuple space scheme ψ that complies with *ABCDE*, i.e. $\psi \in \Psi_{ABCDE}$, but not with *A′ B′ C′ D′ E′*, i.e. $\psi \notin \Psi_{A'B'C'D'E'}$.

3.3 Consequences of the Formalization

Structure on Fields. It is shown that the structure of Ψ_{12} schemes are trees. Except for A=0 or B=0, all schemes can be reduced to them. Therefore, the rest of this contribution lays emphasis on Ψ_{12} schemes.

Theorem 3.3.1. Each element of Ψ_{12} is a tree.

Proof. Assume that there is a $(F,match_F) \in \Psi_{12}$ that is not a tree, i.e.

$$\exists f,f_1,f_2 \in F:\ f \leq_F f_1 \wedge f \leq_F f_2 \wedge \neg\, f_1 \leq_F f_2 \wedge \neg\, f_2 \leq_F f_1 \ .$$

If $f_1,f_2 \in C$, then $f_1 \leq_F class(f)$ and $f_2 \leq_F class(f)$ ensues from the definition of $match_F$. However, $(C,\leq_C)$ is a tree (A=1), so that $\{f_1,f_2\}$ cannot be an anti-chain. If $f_1,f_2 \in I$, then $class(f_1) = class(f) = class(f_2)$ can be followed (B=1). Again, $\{f_1,f_2\}$ cannot be an anti-chain, because $(\kappa,match_\kappa)$ is a tree (B=2). Finally, if $f_1 \in C$ and $f_2 \in I$, then $f_1 \leq_F class(f) = class(f_2)$. Hence, $f_1 \leq_F f_2$ ensues from the transitivity of $match_F$.

Theorem 3.3.2. $(F,match_F) \in \Psi_{A1}$ implies for all $f_1,f_2 \in F$

a) $match_F(f_1,f_2) \rightarrow class(f_1) \leq_C class(f_2)$,
b) $f_1 \sim f_2 \rightarrow class(f_1) = class(f_2)$.

Proof.

a) B=1 implies that $match_F$ is transitive, therefore $match_F(class(f_1),f_2)$. $match_F$ is a matching relation, so that $class(f_1) \leq_C class(f_2)$.
b) Because of the antisymmetry of $\leq_C$ a direct outcome of a).

Therefore, $F^\sim = C^\sim \cup I^\sim$ with

$$C^\sim := \{\ c^\sim \mid c \in C\ \} \quad \text{and} \quad I^\sim_{c^\sim} := \{\ i^\sim \mid i \in I_c\ \}$$

is well-defined for B=1. Then, $\leq_{C^\sim}$ is defined as

$$\leq_{C^\sim} := \{\ (c_1^\sim,c_2^\sim) \in C^{\sim 2} \mid class(c_1) \leq_C class(c_2)\ \} \ .$$

Theorem 3.3.3. $(F,match_F) \in \Psi_{A1}$ implies $(F^\sim,match_{F^\sim}) \in \Psi_{A2}$.

Proof. During partitioning, the class hierarchy is left unchanged, so that the index on the A-axe does not change. The equivalence relation ~ on $F^\sim$ is the identity mapping on $F^\sim$. Therefore, criteria B=1 and B=2 hold for $(F^\sim,match_{F^\sim})$, too.

One Field Tuples. Let $\tau_1(F)$ denote the set of tuples that have only one field, i.e. $\tau_1(F) = F^1$. It is supposed that F and $\tau_1(F)$ are *isomorph* [17], if matching on tuples is induced by matching on fields.

Theorem 3.3.4. $(F,match_F,\Im(F),match_\Im) \in \Psi_{ABCD1}$ and $\tau_1(F) \subset \Im(F)$, then $\tau_1(F)$ is *isomorph* to F.

Proof. Let $f,f^* \in F$ and $\varphi: F \rightarrow \tau_1(F)$ with $\varphi(f) := (f)$. Hence, φ is bijective and $match_\Im(\varphi(f),\varphi(f^*)) \Leftrightarrow \omega(\varphi(f),\varphi(f^*)) \Leftrightarrow match_F(\Pi_1(\varphi(f)),\Pi_1(\varphi(f^*))) \Leftrightarrow match_F(f,f^*)$. Therefore, φ is an isomorphism.

Structure on Tuples. Let $\leq_\Im := match_\Im$ denote the matching relation on tuples, if $match_\Im$ is an order on $\Im(F)$.

Theorem 3.3.5. For $(F,match_F,\Im(F),match_\Im) \in \Psi_{12101}$, $(\Im(F),\leq_\Im)$ is a semilattice.

Proof. In the first step, the theorem is proven for D=1. Since $(F,\leq_F)$ is a tree, it is a semilattice. Hence, $match_F$ is an order on F, so that $match_\Im = \omega$ is an order on $\Im(F)$. Let $s_1,s_2,t_1,t_2 \in \tau(F)$ with $s_j \leq_\Im t_k$ $(j,k \in \{1,2\})$ where $\{s_1,s_2\}$ and $\{t_1,t_2\}$ are anti-chains. If the infimum was not well-defined, there would be no $x \in \tau(F)$ with $s_j \leq_\Im x \leq_\Im t_k$ $(j,k \in \{1,2\})$. The length of the vectors s_1,s_2,t_1,t_2 is identical. Let it be denoted by *n*. Then, $x \in \tau(F)$ defined by

$$|x|=n \text{ and } \forall i \in \{1,...,n\}: \Pi_i(x) = \inf(\Pi_i(t_1),\Pi_i(t_2))$$

leads to a contradiction, because $\Pi_i(s_j) \leq_F \Pi_i(x) \leq_F \Pi_i(t_k)$ with $j,k \in \{1,2\}$ and $i \in \{1,...,n\}$. In the second step (induction step), it is proven that a $(\Im(F),\leq_\Im)$ with the depth of recursion *n+1* is a semilattice, if all $(\Im(F),\leq_\Im)$ with the depth of recursion *n* are a semilattices. This is shown in analogy to step one.

Example. Figure 5 gives an example of ordered sets with one being a semilattice. Note, that $(\Im(F),\leq_\Im)$ is a semilattice in the scheme of Figure 2.

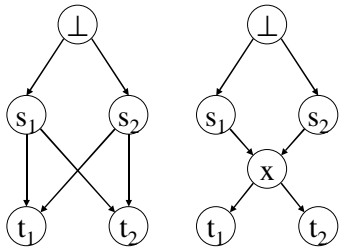


Figure 5. Graphical representation of two ordered sets. The left one is no semilattice, because the infimum of t_1 and t_2 is not well-defined. The right one is a semilattice.

Nested Tuples. As it is clear from their definition, nested tuples are trees with fields as their leaves. For an arbitrary $t \in \tau$, let t^i denote the set of subtrees with the depth *i*, that is

$$t^0 := t \quad \text{and} \quad t^i := \{ \Pi_j(s) \mid s \in t^{i-1} \backslash F \wedge j \in \{1,...,|s|\} \} .$$

Then, the *depth* of *t* is defined as $\text{depth}(t) := \max\{ i \mid t^i \neq \varnothing \}$ and the *breadth* of *t* is defined as $\text{breadth}(t) := \max\{ |s| \mid i \in \{0,...,\text{depth}(d)-1\} \wedge s \in t^i \}$. The definitions are extended to an arbitrary semantic tuple $T \in \Gamma$ by

$$\text{depth}(T) := \max_{t \in T} \{\text{depth}(t)\} \quad \text{and} \quad \text{breadth}(T) := \max_{t \in T} \{\text{breadth}(t)\} .$$

Theorem 3.3.6. Let $(F,match_F,\Im(F),match_\Im) \in \Psi_{ABC01}$ and an arbitrary $\Im' \subseteq \Im$ that complies with $\exists \beta,\theta: \forall T \in \Im' [: \text{breadth}(T) \leq \beta \wedge \text{depth}(T) \leq \theta]$. Then, there exists a set of fields F* and tuples $\Im^*$ with $(F^*,match_{F^*},\Im^*(F^*),match_{\Im^*}) \in \Psi_{ABC11}$ and a mapping $\Phi: \Im' \rightarrow \Im^*$, such that

$$\forall T_1,T_2 \in \Im' : [\ match_\Im(T_1,T_2) \leftrightarrow match_{\Im^*}(\Phi(T_1),\Phi(T_2))\ ] .$$

Proof. F* is obtained from *F* by adding the class Υ with $I_\Upsilon = \varnothing$. There is no subclass of Υ, it is only matched by $\bot_F$. Hence, $(F,match_F)$ and $(F^*,match_{F^*})$ share the same *A* and *B* indices. Let τ^* denote the set of actual and formal tuples with the maximal dimension $d = \beta^\theta$, that are not nested (D=1). At first, the mapping Φ is defined on τ'. Therefore, let $\phi: \tau' \to \tau$ denote the mapping that extends the tree structure of tuples, so that each leaf node is located on the same level. A tuple $t \in \tau'$ given, such a tree is gradually obtained by extending an arbitrary subtree $s \in t^i$ with $i < \theta$ and $|s| < \beta$ to s', such that

$$s = \bot_F \to s' = (\bot_F)^\beta \quad \text{and} \quad s \neq \bot_F \to s' = s \times \Upsilon^{\beta - |s'|}.$$

Figure 6 illustrates how ϕ extends the tree structure. For $t_1, t_2 \in \tau'$, it is inductively shown that $match_\Im(t_1,t_2) \leftrightarrow match_\Im(\phi(t_1),\phi(t_2))$. If all leaves of t_1 and t_2 are on the same level, $\phi(t_1) = t_1$ and $\phi(t_2) = t_2$, hence the equivalence is trivial. If the leaves of t_1 and t_2 are at least at depth *i*, the tuples are extended to t_1' and t_2', so that the leaves of t_1' and t_2' are at least at depth i+1. Two cases may occur while extending the subtrees s_1 and s_2 to s_1' and s_2': If $s_1 = \bot_F$, then $s_1' = (\bot_F)^\beta$ still matches every s_2', since $|s_2'|$. Otherwise, $match_\Im(s_1', s_2') \leftrightarrow match_\Im(s_1,s_2) \wedge |s_1| = |s_2| \leftrightarrow match_\Im(s_1,s_2)$ ensues from the definition of matching on Υ. Hence, the induction hypothesis implies that the equivalence $match_\Im(t_1,t_2) \leftrightarrow match_\Im(\phi(t_1),\phi(t_2))$ holds.

Let $\phi^*: \phi(\tau') \to \tau^*$ denote the mapping that assigns to a tuple the vector of fields, as they are found in a depth-first search. The tuple's fields are located on the same level, so that the dimension of the vector is β^θ. Obviously, $match_\Im(t_1', t_2') \leftrightarrow match_{\Im^*}(\phi^*(t_1'), \phi^*(t_2'))$ ensues from $match_{\Im^*} = \omega$ (E=1). Therefore, Φ is defined on actual and formal tuples by $\phi^* \bullet \phi$, so that $match_\Im(t_1,t_2) \leftrightarrow match_{\Im^*}(\Phi(t_1),\Phi(t_2))$ holds for arbitrary $t_1, t_2 \in \tau'$. Φ is extended to semantic tuples by

$$\forall T \in \Gamma' : \Phi(T) := \bigcup_{t \in \Im} \Phi(t)$$

and $\Gamma^* := \Phi(\Gamma')$. Hence, $\Im'$ and $\Im^*$ share the same *C* index and $match_\Im(T_1,T_2) \leftrightarrow match_{\Im^*}(\Phi(T_1),\Phi(T_2))$ holds for arbitrary $T_1, T_2 \in \Im'$.

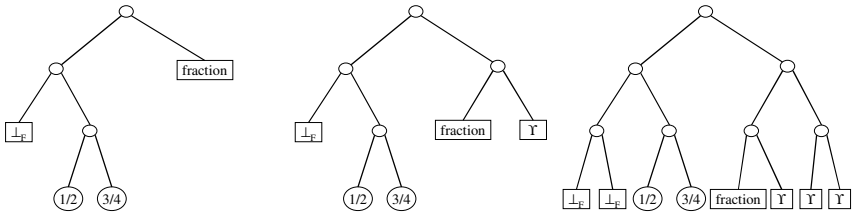


Figure 6. For β=2 and θ=3, the mapping of the tuple $t = ((\bot_F,(1/2,3/4)),fraction)$ to $\phi(t)$ is illustrated. Note, that it is trivial to map $\phi(t)$ to a vector of 2^3=8 dimensions, as it is done by $\Phi(t) = \phi^*(\phi(t)) = (\bot_F,\bot_F,1/2,3/4,fraction,\Upsilon,\Upsilon,\Upsilon)$.

Semilattices. It has been conjectured in [46], that it is a promising starting point for achieving scalability, if $(F,\leq_F)$ and $(\Im(F),\leq_\Im)$ are semilattices [17]. E.g. the tuples which are stored on a server can be represented by their infimum. Then, the

matching of the infimum by a template is a sufficient condition for a successful query on the server.

Let $(L,\leq_L)$ denote a semilattice. Furthermore, let the infimum be denoted by the binary operator $\wedge_L \subseteq L^2$. For an element $l \in L$, its ancestors l^- and successors l^+ are defined by

$$l^- := \{ k \in L \mid k \leq_L l \} \text{ and } l^+ := \{ k \in L \mid l \leq_L k \} .$$

Theorem 3.3.7. $\forall l_1,l_2 \in L: \; l_1^- \cap l_2^- = (l_1 \wedge_L l_2)^-$.

Proof. $k \in l_1^- \cap l_2^- \Leftrightarrow k \leq_L l_1 \wedge k \leq_L l_2 \Leftrightarrow k \leq_L \inf(l_1,l_2) \Leftrightarrow k \in (l_1 \wedge_L l_2)^-$.

3.4 Comparison to Other Formal Approaches

The presented formalization of tuple spaces aims at identifying and exploiting structural restrictions in order to achieve scalability. Hence, the data and its retrieval, i.e. the tuples and templates, is the subject of the formalization. Furthermore, the taxonomy distinguishes different levels of structural restrictions. Therefore, it is possible to depict the domain of scalability concepts, i.e. the tuple space schemes that the concept addresses.

In contrast, the formalization given by Gelernter himself [34] aims at validating Linda implementations. In consequence, the formalization is focused on the semantics of tuple space operations and on the processes that cooperate through the tuple space. According to his formalization, the set of fields is given and tuples are defined as vectors of fields. The set of templates is the power set of tuples. Although this formalization is sufficient for the validation of Linda implementations, it does not suit well as a base for scalability studies. The formalization is a priori bound to Linda tuple space schemes, so that recent extensions of tuple spaces are not taken into account. Furthermore, there is no notion of actual and formal fields and their matching. Consequently, the formalization overestimates the expressiveness of templates in current tuple space implementations by introducing them as semantic tuples. Hence, the formalization partly abstracts from structural restrictions in tuple spaces schemes.

The same is valid for formal approaches that are based on process calculi [11, 45]. These approaches aim at modeling asynchronous coordination and its semantics. Hence, they lay stress on processes and operations, so that they take a narrow view of tuple space schemes. E.g. even though LLinda [45] distinguishes formal and actual tuples, its definition of tuples, fields and matching is identical to the one given in Ψ_{23111} schemes.

In conclusion, previous approaches do not aim at formalizing and capturing the nuances of data and its retrieval in tuple spaces. Therefore, they fail to provide a formal foundation for identifying and exploiting structural restrictions in tuple space schemes.

3.5 Distribution of Tuples

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.

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 $\{1,...,p\}$. In addition, $\Im$ depicts a set of tuples. Let Δ denote the set of mappings $\Im \rightarrow P(\{1,...,p\})\setminus\varnothing$, called distributions.

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

$$\forall T_1,T_2\in\Im: \text{match}_\Im(T_1,T_2) \rightarrow \delta(T_1)\cap\delta(T_2) \neq \varnothing .$$

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. 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

$$\forall T_1,T_2\in\Im: \text{match}_\Im(T_1,T_2) \rightarrow \delta_r(T_1)\cap\delta_w(T_2) \neq \varnothing .$$

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 extension 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 7 illustrates this principle.

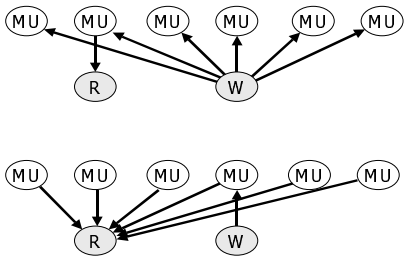


Figure 7. $(\delta^*,\delta_1)\in\Delta_{wr}$ *(above)* and $(\delta_1,\delta^*)\in\Delta_{wr}$ *(below)*. The arrows indicate which servers *(MU)* are taken into account for writing *(W)* and reading *(R)* a tuple.

Dynamic Behaviour. The number of servers should be adjusted to the number of tuples. Hence, a new server has to be added, if several new tuples have to be stored. On the other hand, a server can be merged with another that stores only few tuples due to deletions. Incrementing or decrementing p to $p'=p\pm1$ calls for adjusting the distribution (δ_w,δ_r) to $(\delta_w'\,\delta_r'\,)$. Note, that if a server with the index $p^*<p$ is abandoned, the remaining servers $1,...,p^*-1,p^*+1,...,p$ are reindexed to $1,...,p^*-1,p^*,...,p-1$. If a server is added, replicas of a tuple T have to be stored on $\delta_w'\,(T)\setminus\delta_w(T)$ and to be removed from $\delta_w(T)\setminus\delta_w'\,(T)$. Therefore, the total number of changes is

$$\sum_{T\in\Im}\left|\delta_w'(T)\setminus\delta_w(T)\right|+\left|\delta_w(T)\setminus\delta_w'(T)\right| \ .$$

Hence, the more the distribution is altered, the more it is costly to add a new server.

4 Analysis of Former Approaches towards Scalability

Former studies of scalability in tuple spaces [8, 40, 46] suggest the use of a hash function, in order to compute the distribution. It assigns either one arbitrary server or all servers to a tuple. Hence, this concept 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, is often not solvable.

At first, a formalization of scalability is given. It provides the foundation for the analysis of distributions based on hashing. The deficiencies of other approaches towards scalability have already been shown in section 2.3. This chapter assumes rather restrictive tuple space schemes, in order to present a fair view of former studies. Therefore, semantic and nested tuples are prohibited (C=D=1) and matching is not customized (B=2; E=1).

4.1 A Deterministic Model of Scalability

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

$P(\Im)$ is closed in regard of the complement, intersection and union of its elements, hence $P(\Im)$ is a σ-algebra [27] of sets in $\Im$. Let $\pi_w,\pi_r: P(\Im)\to[0,1]$ denote the mappings, that assign the *frequency* of writing and reading operations to sets of tuples. The mappings have to comply with $\pi_w(\Im)=1=\pi_r(\Im)$. In addition, the mappings ensure complete additivity, i.e. given a countable collection of non-overlapping sets $\Re_i\in P(\Im)$

$$\sum_i \pi_w(\Re_i)=\pi_w(\bigcup_i \Re_i) \quad \text{and} \quad \sum_i \pi_r(\Re_i)=\pi_r(\bigcup_i \Re_i)\,.$$

Hence, π_w and π_r are *probability measures* [27] and the triples $(\Im,P(\Im),\pi_w)$ and $(\Im,P(\Im),\pi_r)$ are *probability spaces* [27]. Let Π_{wr} denote the set of *usage profiles* (π_w,π_r) with π_w and π_r being such probability measures.

For an arbitrary distribution δ, the mapping $\delta^{-1}:\{1,...,p\}\to P(\Im)$ assigns the respective set of tuples to a server, i.e. $\delta^{-1}(q):=\{T\in\Im \mid q\in\delta(T)\}$. In addition, let $\Im_n\subseteq\Im$ denote a multiset of *n* tuples, that are stored in the tuple space. Then, $\Im_n(q)\subseteq\Im_n$ is defined as the multiset of tuples on server *q* by $\Im_n(q):=\Im_n\cap\delta_w^{-1}(q)$.

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,

$$S_M(q):=|\Im_n(q)| \quad \text{and} \quad S_Q(q):=n\cdot\sum_{T\in\delta_r^{-1}(q)}\pi_r(T)\,.$$

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 [32].

$$A_w:=\sum_{i=1}^{p}\pi_w\left(\{T\in\Im \mid |\delta_w(T)|=i\}\right) \quad \text{and} \quad A_R:=\sum_{i=1}^{p}\pi_r\left(\{T\in\Im \mid |\delta_r(T)|=i\}\right)\,.$$

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 $\chi:\{1,...,p\}\times\Im\to\{0,1\}$ 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,

$$\chi(q,T)=1 \;:\Leftrightarrow\; \exists T'\in\Im_n(q):\ match_\Im(T,T')$$

$$\chi(T) := \sum_{q \in \delta_r(T)} \chi_r(q,T) .$$

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

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

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.

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).

It has been suggested [46] that properties in O(log n) may be regarded as scalable, too. Although such an extension is helpful for more complicated distributions, i.e. based on trees, it is not functional for distributions based on hashing.

In the context of chapter 1, $S_M(q)$ and $S_Q(q)$ represent the scalability dimension *(2)* and *(4)* respectively. On the other side, A_w, A_r and A_R are linked with the scalability dimension *(5)*.

Examples. Whatever scheme is used, a scaling property opposes the scaling of another. The example assumes $p \in \Omega(n)$. 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^*), $p \in \Omega(n)$ and a δ_1 that balances the load, $S_M(q)$ 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.

Definition. A tuple space is scalable if and only if all of its properties scale.

Theorem 4.1.1. The following conditions are sufficient conditions for a tuple space not to be scalable:

a) $\forall T \in \Im: |\delta_w(T)| \in \omega(1) \Rightarrow A_w$ does not scale
b) $\exists T^* \in \Im: \pi_r(T^*) \in \omega(1/n) \Rightarrow \forall q \in \delta_r(T^*): S_Q(q)$ does not scale
c) $p \in o(n) \Rightarrow \exists q_1,q_2 \in \{1,...,p\}: S_Q(q_1)$ and $S_M(q_2)$ do not scale
d) Let g denote a surjective mapping on natural numbers.
$\forall T \in \Im: |\delta_w(T)| \in \Omega(g(p)) \wedge p/g(p) \in o(n) \Rightarrow \exists q \in \{1,...,p\}: S_M(q)$ does not scale

Proof.

a) $A_w = \sum_{T \in \Im} \pi_w(T) \cdot \omega(1) = \omega(1) \cdot \sum_{T \in \Im} \pi_w(T) = \omega(1) \notin O(1)$.

b) Let $q^* \in \delta_r(T^*)$, then $S_Q(q^*) \geq n \cdot \pi_r(T^*) \notin O(1)$.

c) Assume $\forall q \in \{1,...,p\}$: $S_Q(q)$ and $S_M(q)$ scale, then $S_Q, S_M \in O(p) = o(n)$ with $S_Q := \sum_{q=1}^{p} S_Q(q)$ and $S_M := \sum_{q=1}^{p} S_M(q)$. But

$$S_Q = n \cdot \sum_{q=1}^{p} \sum_{T \in \delta_r^{-1}(q)} \pi_r(T) \geq n \cdot \sum_{T \in \Im} \pi_r(T) = n \notin o(n) ,$$

$$S_M = \sum_{q=1}^{p} |\{T \in \Im_n \mid q \in \delta_w(T)\}| = \sum_{q=1}^{p} \sum_{\substack{T \in \Im_n \\ q \in \delta_w(T)}} 1 = \sum_{T \in \Im_n} |\delta_w(T)| \geq n \notin o(n).$$

d) Assume, that $S_M(q)$ scales for every $q \in \{1,...,p\}$, then $S_M \in O(p)$. But

$$S_M = \sum_{T \in \Im_n} |\delta_w(T)| = n \cdot \Omega(g(p)) = \omega(p/g(p)) \cdot \Omega(g(p)) \in \omega(p) .$$

Part *(a)* proves that writing operations do not scale, if every tuple has to be replicated on more than a fixed number of servers. Part *(b)* states that the number of queries on a subset of servers does not scale, if a template is used with a frequency $\omega(1/n)$. Furthermore, part *(c)* shows that the number of servers has to be kept at least proportional to the number of tuples. All these results make sense and thus partially justify the model. In addition, part *(d)* relates the degree of replication to the number of servers needed. E.g. if every tuple is to be replicated to $\sqrt{p}$ servers, then the number of servers has to be at least n^2. Hence, replication to a large number of servers is virtually impossible, if scalability is to be achieved.

4.2 Distribution Based on Hash Codes

The hash function $h: \Im \rightarrow Nat$ is only partially defined. Let $Def(h) \subseteq \Im$ denote the set of tuples with a defined hash code. Such a tuple is assigned to exactly one server. It has been suggested [8] that such a tuple *T* is assigned to the server indexed

$$1 + [h(T) \bmod p] .$$

Therefore, the appropriate server can be determined in O(1). However, if the number of servers is changed, most of the tuples have to be moved to another server. Therefore, adjusting the number of servers to the number of tuples is costly, i.e. O(n). Another concept [40] avoids this problem, but it takes O(log p) to determine the appropriate server of a tuple.

Tuples without a defined hash code are assigned to all servers by the distribution. E.g. a query with the template $\perp_{\Im}$ has to be performed on all servers. In conclusion, the hash function *h* induces a distribution δ_h with

$$\forall T \in \Im: |\delta_h(T)| \in \{1,p\} \wedge \forall T,T' \in \Im: [\, h(T)=h(T') \rightarrowtail \delta_h(T)=\delta_h(T') \,] .$$

In [8, 40], it is not foreseen that a tuple T with $\delta_h(T) = \{1,...,p\}$ is written to the tuple space. Hence, the following assumes that a tuple T with $\delta_h(T) = \{1,...,p\}$ is written to only one arbitrary server. Therefore, (δ_{h1},δ_h) is the write/read distribution used in this concept with

$$\forall T \in \Im: |\delta_{h1}(T)| = 1 \;\wedge\; \delta_{h1}(T) \subseteq \delta_h(T) .$$

4.3 Analysis of the Concept

The concept, as introduced in the previous section, comes with several drawbacks. In this section, the consequences of its approach are analyzed.

Correctness. The write/read distribution has to be a permissible, of course, so that $(\delta_{h1},\delta_h) \in \Delta_{wr}$. Therefore,

$$\forall T_1,T_2 \in \mathrm{Def}(h): \mathrm{match}_{\Im}(T_1,T_2) \rightarrow \delta_h(T_1)=\delta_h(T_2) .$$

Since this has to be true for all *p*, the hash function has to comply with

$$\forall T_1,T_2 \in \mathrm{Def}(h): \mathrm{match}_{\Im}(T_1,T_2) \rightarrow h(T_1)=h(T_2) .$$

Furthermore, a tuple T_1 without a hash code may not be matched by a tuple T_2 with a hash code. This is due to $|\delta_{h1}(T_1)| = 1 = |\delta_h(T_2)|$ and an arbitrary $\delta_{h1}(T_1)$, which opposes $\delta_{h1}(T_1) \cap \delta_h(T_2) \neq \varnothing$. Hence,

$$\forall T_1,T_2 \in \Im: \mathrm{match}_{\Im}(T_1,T_2) \rightarrow [T_1 \notin \mathrm{Def}(h) \vee h(T_1)=h(T_2)] .$$

Hash Functions. Let $H_{\Im}$ denote the set of correct hash functions on $\Im$. The hash function h^{σ} has been suggested [8] for schemes in Ψ_{23111}, i.e. Linda schemes. h^{σ} is based on hashes on signatures, so that

$$\mathrm{Def}(h^{\sigma}) = \{t \in \tau(F) \mid \forall i \in \{1,...,|t|\}: \textstyle\prod_i(t) \neq \perp_F\} \text{ and } \forall t \in \mathrm{Def}(h^{\sigma}): h^{\sigma}(t) = h^{\sigma}(\sigma(t)) .$$

For this kind of schemes, h^{σ} is correct, i.e. $h^{\sigma} \in H_{\Im}$. In addition, the concept of distribution based on hash functions is hidden from the application programmer, if h^{σ} is used. However, h^{σ} ceases to be correct for C=2 or D=0. Even worse, it is proven that the tuple space does not scale in practice.

Theorem 4.3.1. For $t^{\sigma} \in \tau^{\sigma}$ with $|t^{\sigma} \cap \Im_n| \in \omega(1)$ and $\delta_h(t) = \{q\}$

a) $S_M(q)$ does not scale

b) $\sum_{t \in t^{\sigma}} \pi_r(t) \in \omega(1/n) \Rightarrow S_Q(q)$ does not scale

Proof.

a) $S_M(q) \geq |t^{\sigma} \cap \Im_n(q)| = |t^{\sigma} \cap \Im_n| \in \omega(1)$

b) $S_Q(q) \geq n \cdot \sum_{t \in t^{\sigma}} \pi_r(t) \in \omega(1)$

In contrast, the hash function introduced in [46] is only defined on the set of actual tuples. Hence, formal templates cause O(p) for reading operations.

In order to fit to a specific usage profile, the hash function is customizable in [40]. Hence, the application programmer has to define the hash function by himself. In addition, he has to ensure that his hash function is correct and that the tuple space is scalable for his profile. The nature of these constraints is examined in the following sections.

Constraints on Correctness. Let h denote a hash function on the scheme $(F,match_F,\Im(F),match_\Im)\in\Psi_{12111}$. Hence, $\Im$ is a semilattice.

Theorem 4.3.2. For arbitrary $T_1,T_2\in\Im$, these are necessary conditions for $h\in H_\Im$:

a) $T_1\in Def(h) \Rightarrow \forall T\in T_1^+: h(T) = h(T_1)$
b) $T_1\notin Def(h) \Rightarrow \forall T\in T_1^-: T\notin Def(h)$
c) $T_1,T_2\in Def(h) \wedge h(T_1) \neq h(T_2) \Rightarrow T_1^+\cap T_2^+ = \varnothing$
d) $h(T_1) \neq h(T_2) \Rightarrow \forall T\in T_1^-\cap T_2^-: T\notin Def(h)$

Proof.

a) $T_1\leq T \Rightarrow T_1\notin Def(h) \vee h(T_1)=h(T) \Rightarrow h(T_1)=h(T)$
b) $T\leq T_1 \Rightarrow T\notin Def(h) \vee h(T)=h(T_1) \Rightarrow T\notin Def(h)$
c) Assume $T\in T_1^+\cap T_2^+$, then (a) concludes in $h(T_1)=h(T)=h(T_2)$.
d) $T\leq T_1 \wedge T\leq T_2 \Rightarrow T\notin Def(h) \vee h(T_1)=h(T)=h(T_1) \Rightarrow T\notin Def(h)$

Keys. Let τ_n denote the set of tuples that have *n* fields. This set is partitioned by $J\subseteq\{1,...,n\}$ into sets t^J of tuples that are identical on the position $j\in J\subseteq\{1,...,n\}$. Hence,

$$t^J := \{\ t'\in\tau_n \mid \forall j\in J: \Pi_j(t) = \Pi_j(t'\ )\ \}\ .$$

Then, the set *K* is a *key* to a given *n* if and only if

$$\forall t\in\tau_n: \forall t_1,t_2\in t^K: [h(t_1)=h(t_2) \vee t_1,t_2\notin Def(h)].$$

In other words, for the computation of the hash function it is enough to take into account the fields of the key. Note, that $\{1,...,n\}$ is a key. A key is called a *minimal key* if and only if it has no subset that is a key, too.

Theorem 4.3.3. For an arbitrary *n* and $h\in H_\Im$

a) there is exactly one minimal key.
b) all keys are supersets of the minimal key.

Proof.

a) Assume two different minimal keys K_1 and K_2. Hence, $K'=K_1\cap K_2$ is no minimal key, so that there have to be tuples $t_1\in\tau_n$ and $t_2\in t_1^{K'}$ with

$$h(t_1) \neq h(t_2) \wedge [t_1\in Def(h) \vee t_2\in Def(h)]\ .$$

Let $t\in\tau_n$ denote a tuple with

$$\forall m \in \{1,2\}: \forall j \in K_m: \Pi_j(t) = \Pi_j(t_m) .$$

Then $t_m \in t^{Km}$ and $t \in Def(h) \leftrightarrow t_m \in Def(h)$, so that $t_1, t_2 \in Def(h)$. Hence, $h(t_1) = h(t) = h(t_2)$.

b) Let K^* denote the minimal key. Assume that there is a key K_1 that is not a superset of K^*, so that $K^* \setminus K_1 \neq \emptyset$. Therefore, the minimal key K_2 with $K_2 \subseteq K_1$ is not the same as K^*, thus contradicting (a).

In other words, the hash function is only allowed to take into account a specific set of fields.

Scalability. While defining a correct hash function, the application programmer has to bear in mind that the tuple space is scalable for his profile. Whatever hash function is used, A_w scales.

Theorem 4.3.4. The following conditions are sufficient conditions that a tuple space is not scalable:

a) $\exists T \in Def(h): |\Im_n \cap T^+| \in \omega(1) \Rightarrow \forall q \in \delta_h(T): S_M(q)$ does not scale

b) $\sum_{T \notin Def(h)} \pi_r(T) \in \omega(1/n) \Rightarrow \forall q \in \{1,...,p\}: S_Q(q)$ and A_R do not scale

c) Let g denote a surjective mapping on natural numbers.
$p \in \Omega(n) \wedge \exists \Im^* \subseteq \Im \setminus Def(h): \sum_{T \in \Im^*} \pi_r(T) \in \omega(g(n)/n) \wedge | \bigcup_{\substack{t \in \Im_n \cap T^+ \\ T \in \Im^*}} \delta_{h1}(t) | \in O(g(n))$

$\Rightarrow A_r$ does not scale

Proof.

a) Theorem 4.3.2*(a)* implies $S_M(q) \geq |\Im_n(q) \cap T^+| = |\Im_n \cap T^+| \in \omega(1)$.

b) $S_Q(q) \geq n \cdot \sum_{\substack{T \notin Def(h) \\ q \in \delta_h(T)}} \pi_r(T) = n \cdot \sum_{T \notin Def(h)} \pi_r(T) \in \omega(1)$.

$A_R \geq \sum_{T \notin Def(h)} \pi_r(T) \cdot |\delta_h(T)| = n \cdot \sum_{T \notin Def(h)} \pi_r(T) \in \omega(1)$.

c) Let $T \in \Im^*$, then $\chi(T) = | \bigcup_{t \in \Im_n \cap T^+} \delta_{h1}(t) | \in O(g(p))$. Hence,

$$A_r \geq \sum_{T \in \Im^*} \pi_r(T) \cdot \frac{|\delta_r(T)|}{\max(1, \chi(T))} = \sum_{T \in \Im^*} \pi_r(T) \cdot \frac{p}{O(g(n))} = \Omega(n/g(n)) \cdot \sum_{T \in \Im^*} \pi_r(T)$$

$\in \omega(1)$.

Part *(a)* proves that there are only O(1) tuples that are matched by a template with a hash code. Part *(b)* states that profiles are only allowed, if the total frequency of templates without hash code takes O(1/n). Finally, part *(c)* shows that, a template without hash code given, the quotient

$$\frac{\text{number of different hash codes of the tuples matched by the template}}{\text{frequency of reading operations with the template}}$$

has to be at least proportional to *n*.

4.4 Scenario

Theorem 4.3.4*(b)* assumes that bulk operations should scale, too. This is an arguable assumption, hence this section will not make use of it. Beyond that, the scenario' s scheme Ψ_{23111}) is by far more structured than schemes found in practice. In spite of this, the following scenario clearly demonstrates that the definition of an appropriate hash function is a highly non trivial task.

Assume that a tuple space is applied for brokering services. These services are provided by hardware devices that are characterized by the tuple `(device_type, device_attributes, device_address)`, e.g. `(printer, 600dpi, 129.13.65.1)`. Therefore, an application may specify the type of device needed and a tuple is returned that indicates the address of an appropriate device. The number of different device types is expected at o(n). Then Theorem 4.3.4*(a)* shows that the device type is not the key of the hash function. Assume that all fields together are used as key, as it is illustrated in Figure 8*(a)*. Theorem 4.3.2*(d)* implies that no template would have a hash value, hence the distribution degenerates to (δ_1,δ^*). Then Theorem 4.3.4*(c)* shows that a template used with the frequency of *z* has to match $n \cdot z$ tuples in the tuple space. By way of contrast, assume that the device type and attributes are used as key, as it is illustrated in Figure 8*(b)*. Then Theorem 4.3.4*(a)* implies that the number of different device types and attributes is proportional to *n*. Furthermore, Theorem 4.3.4*(c)* and Theorem 4.3.2*(d)* show that there have to be at least $n \cdot z$ tuples of a certain device type, if its attributes are not specified by templates with the frequency of *z*.

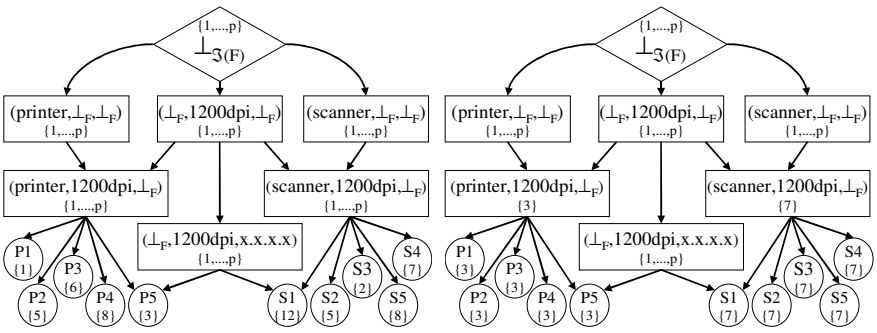


Figure 8. An excerpt of the graph $(\Im,\text{match}_\Im)$ in a service brokering scenario. In the left *(a)*, the device address is part of the key, in the right *(b)* it is not. The set $\delta_h(T)$ is included for every tuple *T*. P1-4/S2-5 depicts 1200dpi printers/scanners with an arbitrary address. P5/S1 is a 1200dpi printer/scanner with the same address x.x.x.x. Note, that Theorem 4.3.2*(d)* implies that the tuple $(\perp_F,1200\text{dpi},x.x.x.x)$ has no hash code.

In a distribution based on hashing [40, 8], a tuple is assigned to all servers or to exactly one. Therefore, the concept introduced in this chapter seems to be too coarse for practical use. However, it is a good starting point for further refinement.

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 $(\Im,match_\Im)$ 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 [8, 40].

The structure of $(\Im,match_\Im)$ is implied by the matching on tuples. Therefore, an arbitrary $match_\Im$ (E=0) 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 E=1. 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. For the moment, it is assumed that tuples may not be nested (D=1).

This chapter introduces a new concept for scalability that has been proposed in [48]. It fully exploits the structure of tuples and 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

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

Assume that $\iota_\Im\colon \Im \to J(Nat)$ maps a tuple to an interval of natural numbers. In addition, $\iota_\Im$ has to comply with

$$\forall T_1,T_2\in\Im\colon match_\Im(T_1,T_2) \;\rightarrow\; \iota_\Im(T_1)\cap\iota_\Im(T_2)\neq\varnothing\ .$$

Note, that ι_h complies with this demand, if it is the generalization of a hash function $h\in H_\Im$, i.e.

$$\forall T\in Def(h)\colon \iota_h(T):=[h(T),h(T)] \;\wedge\; \forall T\notin Def(h)\colon \iota_h(T):=[0,\infty]\ .$$

Furthermore, assume that $\iota_\Im$ complies with the inversion, that is

$$\forall T_1,T_2 \in \Im: \iota_\Im(T_1) \cap \iota_\Im(T_2) \neq \varnothing \;\rightarrow\; T_1 \sim T_2 \,.$$

In general, ι_h does not comply with this demand. E.g. in Figure 8*(b)* (printer,$\perp_F$,$\perp_F$) and (scanner,$\perp_F$,$\perp_F$) do not match, but the cardinalilty of their intersection is *p*. Even though ($\Im$,match$_\Im$) in Figure 8*(b)* is a semilattice, it is shown that there is no mapping $\iota_\Im$ that fulfills this demand.

Assume that there was such a $\iota_\Im$. Then,

$$\iota_\Im(T_j) \cap \iota_\Im(T_k) = \varnothing = \iota_\Im(S_j) \cap \iota_\Im(S_k) \;\text{ with } j,k \in \{1,2,3,4\} \text{ and } j \neq k$$

in Figure 9*(a)*. If $\iota_\Im(T_1) < \iota_\Im(T_2) < \iota_\Im(T_3) < \iota_\Im(T_4)$, then $\iota_\Im(S_1) < \iota_\Im(S_2) < \iota_\Im(S_3)$, too. Therefore, $\iota_\Im(T_1) < \iota_\Im(S_2) < \iota_\Im(T_4)$, so that there is no valid value for $\iota_\Im(S_4)$, because $\iota_\Im(S_2) \subset \iota_\Im(S_4)$.

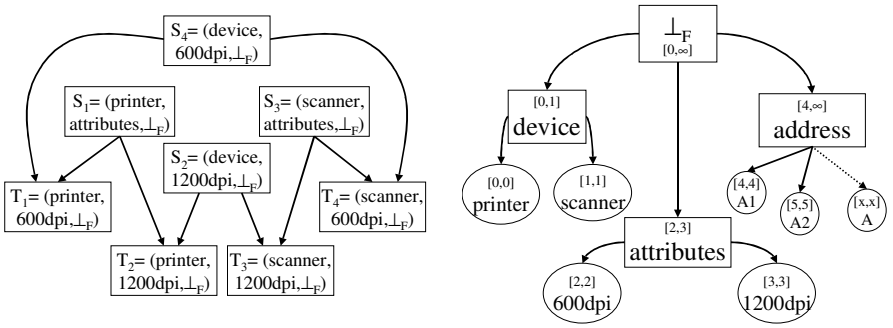


Figure 9. Excerpts of the graph ($\Im$,match$_\Im$) on the left *(a)* and (F,match$_F$) on the right *(b)* in a service brokering scenario. Note, that the definition of a mapping to intervals is trivial, if the graph is a tree as in *(b)*.

However, Figure 9*(b)* suggests that it is no problem to map fields to intervals. This is due to the tree structure of (F,match$_F$). In practice, fields are often structured as trees. E.g. all elements of Ψ_{12} are trees.

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 $\iota_F: F \rightarrow J(Nat)$ that comply with

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

Furthermore, let $I_F^{\subset} \subseteq I_F$ denote a subset of mappings $\iota_F \in I_F$ that in addition comply with

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

$I_F^{\subset}$ and I_F are not empty, because arbitrary constant mappings are element of $I_F^{\subset}$. However, sounder mappings can be defined, if $(F,match_F)$ is a tree, as it is true for elements in Ψ_{12}.

Let $\upsilon: P(S^d) \to S^d$ denote a function that maps a set of hypercubes to the smallest hypercube superset of their union, that is

$$\forall j \in \{1,...,d\}: \Pi_j(\upsilon(\{s_1,....,s_n\})) = [\min(\Pi_j(\bigcup_{k=1}^{n} s_k)), \max(\Pi_j(\bigcup_{k=1}^{n} s_k))] .$$

Theorem 5.2.1. For $U=\{u_1,...,u_n\}$, $V=\{v_1,...,v_m\}$ and $u_j,v_j \in S^d$, it is

a) $\bigcup_{k=1}^{n} u_k \subseteq \upsilon(U)$

b) $\upsilon(U) \cup \upsilon(V) \subseteq \upsilon(U \cup V)$

c) $U \subseteq V \rightarrow \upsilon(U) \subseteq \upsilon(V)$.

Proof.

a) Trivial.

b) For $j \in \{1,...,d\}$, let $(U \cup V)_j = \Pi_j(\bigcup_{k=1}^{n} u_k \cup \bigcup_{k=1}^{m} v_k)$, $U_j = \Pi_j(\bigcup_{k=1}^{n} u_k)$ and $V_j = \Pi_j(\bigcup_{k=1}^{m} v_k)$. Then $U_j \subseteq (U \cup V)_j$ and $V_j \subseteq (U \cup V)_j$, so that $\Pi_j(\upsilon(U)) \subseteq \Pi_j(\upsilon(U \cup V))$ and $\Pi_j(\upsilon(V)) \subseteq \Pi_j(\upsilon(U \cup V))$.

c) Let U' with $U \cup U' = V$, then $\upsilon(U) \subseteq \upsilon(U) \cup \upsilon(U') \subseteq \upsilon(U \cup U') = \upsilon(V)$ as implied by b).

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

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

$$\forall T \in P(\tau(F)) \setminus \{\varnothing\}: \iota_\Im(T) = \upsilon(\bigcup_{t \in T} \iota_\Im(t)) .$$

Therefore, tuples and sets of tuples are mapped to hypercubes with *d* dimensions. Note, that semantic tuples are sets of tuples, so that they fit into the above definition. E.g. for the mapping ι_F of Figure 9*(b)*, it is $I_\Im(\iota_F)(\perp_\Im) = [0,\infty] \times [-1,\infty]^2$ and

$$I_\Im(\iota_F)\ (\ (printer,attributes,address)\ ) = [0,0] \times [2,3] \times [4,\infty] .$$

Theorem 5.2.2 and Theorem 5.2.3 examine the correlation of hypercubes to the matching of tuples.

Theorem 5.2.2. For $(F,match_F,\Im,match_\Im) \in \Psi_{ABC11}$ and $\iota_\Im = I_\Im(\iota_F)$ with $\iota_F \in I_F$, it is

a) $\forall t_1,t_2 \in \tau: match_\Im(t_1,t_2) \rightarrow \iota_\Im(t_1) \cap \iota_\Im(t_2) \neq \varnothing$

b) $\forall T_1,T_2 \in P(\tau)$: $\exists (t_1,t_2) \in T_1 \times T_2$: $match_\Im(t_1,t_2) \rightarrow \iota_\Im(T_1) \cap \iota_\Im(T_2) \neq \varnothing$

c) $\forall t_1,t_2 \in \Im_P$: $\omega_P(t_1,t_2) \rightarrow \iota_\Im(t_1) \cap \iota_\Im(t_2) \neq \varnothing$

d) $\forall t_1,t_2 \in \Im$: $match_\Im(t_1,t_2) \rightarrow \iota_\Im(t_1) \cap \iota_\Im(t_2) \neq \varnothing$.

Proof. Since D=1 and E=1, it is $match_\Im = \omega$ and tuples are not nested.

a) $|t_1| = n = |t_2|$ and it is $match_F(\Pi_j(t_1),\Pi_j(t_2))$ for an arbitrary j with $1 \leq j \leq n$. Therefore, $\iota_F(\Pi_j(t_1)) \cap \iota_F(\Pi_j(t_2)) \neq \varnothing$ and it follows $\iota_\Im(t_1) \cap \iota_\Im(t_2) \neq \varnothing$.

b) $match_\Im(t_1,t_2)$ and a) implies $\iota_\Im(t_1) \cap \iota_\Im(t_2) \neq \varnothing$. Theorem 5.2.1(a) and Theorem 5.2.1(c) imply $\iota_\Im(T_1) \cap \iota_\Im(T_2) \neq \varnothing$.

c) Let $T_1 = \gamma(t_1)$ and $T_2 = \gamma(t_2)$, hence $T_1,T_2 \in P(\tau) \setminus \{\varnothing\}$, so that $\iota_\Im(T_1) = \iota_\Im(t_1)$ and $\iota_\Im(T_2) = \iota_\Im(t_2)$. Then $\omega_P(t_1,t_2)$ implies that there are $(t_1', t_2') \in T_1 \times T_2$ with $\omega_\tau(t_1', t_2')$, so that $\iota_\Im(T_1) \cap \iota_\Im(T_2) \neq \varnothing$ ensues from b).

d) Because of $\Im \subseteq \Im_P$ and $\omega \subseteq \omega_P$ the direct outcome of c).

Theorem 5.2.3. For $(F,match_F,\Im,match_\Im) \in \Psi_{ABC11}$ and $\iota_\Im = I_\Im(\iota_F)$ with $\iota_F \in I_F^{\subset}$, it is

a) $\forall t_1,t_2 \in \tau$: $match_\Im(t_1,t_2) \rightarrow \iota_\Im(t_2) \subseteq \iota_\Im(t_1)$

b) $\forall t_1,t_2 \in \tau$: $[\, |t_1|=n=|t_2| \wedge \forall j \in \{1,...,n\}: \Pi_j(\iota_\Im(t_2)) \subset \Pi_j(\iota_\Im(t_1)) \,] \rightarrow match_\Im(t_1,t_2)$

c) $\forall t_1,t_2 \in \Im_P$: $\omega_P(t_1,t_2) \rightarrow \iota_\Im(t_2) \subseteq \iota_\Im(t_1)$

d) $\forall t_1,t_2 \in \Im$: $match_\Im(t_1,t_2) \rightarrow \iota_\Im(t_2) \subseteq \iota_\Im(t_1)$.

Proof. Since D=1 and E=1, it is $match_\Im = \omega$ and tuples are not nested.

a) If $match_\Im(t_1,t_2)$, then $|t_1| = n = |t_2|$ and it is $match_F(\Pi_j(t_1),\Pi_j(t_2))$ for an arbitrary j with $1 \leq j \leq n$. Therefore, $\iota_F(\Pi_j(t_2)) \subseteq \iota_F(\Pi_j(t_1))$ and it follows $\iota_\Im(t_2) \subseteq \iota_\Im(t_1)$.

b) $\Pi_j(t_2)) \subset \Pi_j(t_1)$ implies $match_F(\Pi_j(t_1),\Pi_j(t_2))$, so that $match_\Im(t_1,t_2)$ is true.

c) Let $T_1 = \gamma(t_1)$ and $T_2 = \gamma(t_2)$, hence $T_1,T_2 \in P(\tau) \setminus \{\varnothing\}$, so that $\iota_\Im(T_1) = \iota_\Im(t_1)$ and $\iota_\Im(T_2) = \iota_\Im(t_2)$. Let $t_2' \in T_2$, then $\omega_P(t_1,t_2)$ implies that there is a $t_1' \in T_1$ with $\omega_\tau(t_1', t_2')$, so that $\iota_\Im(t_2') \not\subseteq \iota_\Im(t_1')$ ensues from a). Therefore, $\iota_\Im(T_2) \subseteq \iota_\Im(T_1)$ is implied by Theorem 5.2.1(a) and

$$\bigcup_{t_2' \in T_2} \iota_\Im(t_2') \subseteq \bigcup_{t_1' \in T_1} \iota_\Im(t_1') .$$

d) Because of $\Im \subseteq \Im_P$ and $\omega \subseteq \omega_P$ the direct outcome of c).

Even if mappings in $I_F^{\subset}$ had to ensure the equivalence of field matching and interval inclusion, $I_\Im$ would not induce an equivalence of tuple matching and hypercube inclusion. E.g. for $\iota_\Im(t_1) = [0,2] \times [0,0]$, $\iota_\Im(t_2) = [0,2] \times [1,2]$ and $\iota_\Im(t_3) = [1,1] \times [0,1]$, it is $\iota_\Im(T) = [0,2] \times [0,2]$ with $T = \{t_1,t_2\}$. Then, $\Pi_j(\iota_\Im(t_3)) \subset \Pi_j(\iota_\Im(T))$ for $j \in \{1,2\}$, but $match_\Im(T,t_3)$ is false. Therefore, the definition of I_F and $I_F^{\subset}$ is confined to establish a necessary condition for the matching of fields.

For an arbitrary set of tuples $\Im' \subseteq \Im$ and $\iota_F \in I_F$, let $I_\Im(\iota_F)(\Im')$ be defined as $I_\Im(\iota_F)(\gamma_P(\Im'))$. Hence, Theorem 5.2.1(b) implies

$$\bigcup_{T \in \Im'} I_\Im(\iota_F)(T) \subseteq I_\Im(\iota_F)(\Im') .$$

5.3 Distribution Based on Hypercubes

The transformation of tuples to hypercubes abstracts from tuples, however without ignoring the structure of tuples, as it is induced by matching. Hence, the tuples may be distributed based on their hypercubes, which gives more room for differing distribution strategies. One strategy maps a hypercube to a set of natural numbers which induces distribution as shown in chapter 4. Another approach assigns to every server a hypercube that identifies its *tuple domain*. The approach introduces adaptivity into the distribution, since it takes into consideration, which tuples are stored in the tuple space. E.g. tuple domains are partitioned in regard of the usage profile.

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

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

with $\{p_1,p_2,...\}$ depicting the set of prime numbers. Then, the assignment $\delta_G \in \Delta$ of a tuple to a set of servers is analogous to section 4.1. E.g. based on [8], it is

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

Theorem 5.3.1. δ_G is a permissible distribution.

Proof. If $match_\Im(T_1,T_2)$, Theorem 5.2.2(d) shows $I_\Im(\iota_F)(T_1) \cap I_\Im(\iota_F)(T_2) \neq \varnothing$. Then, there is an $x \in G(I_\Im(\iota_F)(T_1)) \cap G(I_\Im(\iota_F)(T_2))$. Hence, $\delta_G(T_1) \cap \delta_G(T_2) \neq \varnothing$.

Figure 10 illustrates this concept. Note, that it is the generalization of the distribution in chapter 4, since they are identical in case of

$$\forall T \in \Im: [\, |I_\Im(\iota_F)(T)|=1 \vee I_\Im(\iota_F)(T)=I_\Im(\iota_F)(\bot_\Im) \,] .$$

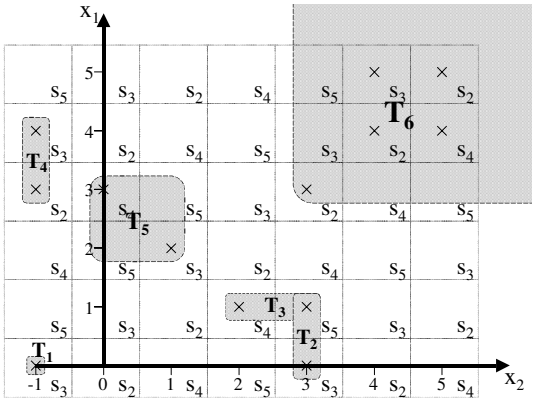


Figure 10. Distribution strategy based on hashing hypercubes for five servers $\{s_1,...,s_5\}$. The example shows tuples with one or two dimensions that are mapped to rectangles as induced by ι_F of Figure 9*(b)*. The displayed tuples are T_1=(printer), T_2=(device,1200dpi), T_3=(scanner,attributes) , 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 $\delta_G(T_6)=\{2,3,4,5\}$.

However, this distribution strategy has to be refined, because $\delta_G(T)$ takes $O(|I_\Im(\iota_F)(T)|)$ in computation complexity and, for an arbitrary mapping G, $|\delta_G(T)|$ takes $O(|I_\Im(\iota_F)(T)|)$, 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.

Tuple Domains. For each server with the index q, Σ_q denotes its tuple domain, i.e. its hypercube. Furthermore, the union of the servers' tuple domains has to be a superset of the hypercubes of the tuples that are stored in the tuple space. Tuple domains are complete, if their union is equal to the considered hyperspace. Tuple domains are disjoint, if they are pairwise disjoint. E.g. complete and disjoint tuple domains have to comply with

$$\bigcup_{T\in\Im_n} I_\Im(\iota_F)(T) \subseteq \bigcup_{q=1}^{p}\Sigma_q = [0,\infty]\times[-1,\infty]^{d-1} \quad \text{and} \quad \forall q,q'\in\{1,...,p\}: \Sigma_q\cap\Sigma_{q'} = \varnothing .$$

The servers' hypercubes induce the distribution $\delta_\Sigma\in\Delta$ with

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

For a given $\delta_\Sigma\in\Delta$, there are arbitrary distributions $\delta_{\Sigma,1}\in\Delta$ that comply with

$$\forall T\in\Im: [\, \delta_{\Sigma,1}(T)\subseteq\delta_\Sigma(T) \;\wedge\; |\delta_{\Sigma,1}(T)| = 1 \,] .$$

Note, that $\delta_{\Sigma,1}$ is not completely defined, in order to be adjustable to other considerations later in this chapter.

Theorem 5.3.2.

a) $\iota_F \in I_F \rightarrow \delta_\Sigma$ is a permissible distribution.

b) $\iota_F \in I_F^{\subset} \rightarrow (\delta_{\Sigma,1}, \delta_\Sigma)$ is a permissible write/read distribution.

Proof.

a) If $match_\Im(T_1,T_2)$, Theorem 5.2.2(d) shows that there is a $x \in I_\Im(\iota_F)(T_1) \cap I_\Im(\iota_F)(T_2)$. Then, $I_\Im(\iota_F)(T_1) \cap I_\Im(\iota_F)(T_2) \subseteq \bigcup_{q=1}^{p} \Sigma_q$ implies that there is a q with $x \in \Sigma_q$. Hence, $q \in \delta_\Sigma(T_1) \cap \delta_\Sigma(T_2)$, so that $\delta_\Sigma \in \Delta_p$.

b) If $match_\Im(T_1,T_2)$, then Theorem 5.2.3(d) shows that $I_\Im(\iota_F)(T_2) \subseteq I_\Im(\iota_F)(T_1)$. Then, $\delta_\Sigma(T_2) \subseteq \delta_\Sigma(T_1)$, so that $\delta_{\Sigma,1}(T_2) \subseteq \delta_\Sigma(T_1)$ ensues from $\delta_{\Sigma,1}(T_2) \subseteq \delta_\Sigma(T_2)$. Hence, $(\delta_{\Sigma,1}, \delta_\Sigma) \in \Delta_{wr}$.

The distribution strategy for complete and disjoint tuple domains is illustrated in Figure 11. Note, that $I_\Im(\iota_F)(T) \cap \Sigma_q = \varnothing$ implies, that server q does not hold any tuple that is matched by template T.

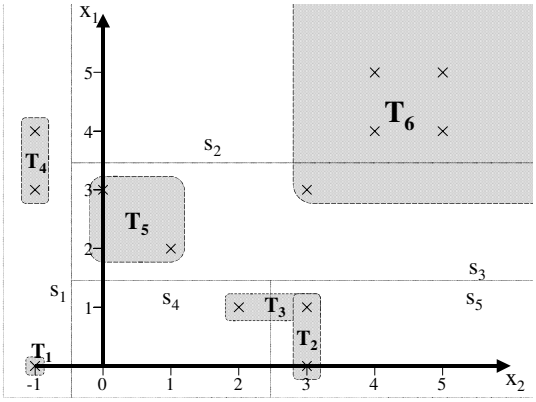


Figure 11. Distribution strategy based on tuple domains that are complete and disjoint. The tuples, ι_F and $\{s_1,...,s_5\}$ are the same as in Figure 10. The servers' rectangles are $\Sigma_1=I_\Im(\iota_F)((\perp_F))$, $\Sigma_2=I_\Im(\iota_F)((address,\perp_F))$, $\Sigma_3=I_\Im(\iota_F)((attributes,\perp_F))$, $\Sigma_4=I_\Im(\iota_F)(\{(device,device),(device,600dpi)\})$ and $\Sigma_5=I_\Im(\iota_F)(\{(device,1200dpi),(device,address)\})$. 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 $\delta_\Sigma(T_6)=\{2,3\}$.

Unlike the strategy based on hashing, 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_Σ, 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.

In order to achieve complete and disjoint tuple domains, a global partitioning scheme has to be applied. Hence, the tuple domain of a server is not deducible

from the tuples that it stores. In general, the server's tuple domain is neither a subset nor a superset of the hypercube of a stored tuple. In addition, for $\iota_F \notin I_F^{\subset}$ tuples have to be stored on every server with an intersecting tuple domain, so that splits may be ineffective.

Alternatively, a server's tuple domain may be deduced from the tuples it stores, i.e.

$$\Sigma_q := \upsilon\Big(\bigcup_{T \in \Im_n(q)} I_{\Im}(\iota_F)(T) \Big) .$$

As a result, tuple domains are neither complete nor disjoint any more. Figure 12 illustrates such tuple domains. As Theorem 5.3.3 shows, they render $(\delta_{\Sigma,1}, \delta_\Sigma)$ a permissible write/read distribution even for $\iota_F \notin I_F^{\subset}$. However, data retrieval includes several servers in general.

Theorem 5.3.3. If tuple domains are not disjoint, i.e. overlapping, such that $\forall T \in \Im_n(q): I_{\Im}(\iota_F)(T) \subseteq \Sigma_q$, then $(\delta_{\Sigma,1}, \delta_\Sigma)$ is a permissible write/read distribution.

Proof. If $match_{\Im}(T^*,T)$ with $T \in \Im_n(q)$, Theorem 5.2.2(d) shows that there is a $x \in I_{\Im}(\iota_F)(T^*) \cap I_{\Im}(\iota_F)(T) \subseteq \Sigma_q$. Then, $I_{\Im}(\iota_F)(T^*) \cap \Sigma_q \neq \varnothing$, so that $\delta_{\Sigma,1}(T) \subseteq \delta_\Sigma(T^*)$ ensues from $q \in \delta_\Sigma(T^*)$. Hence, $(\delta_{\Sigma,1}, \delta_\Sigma) \in \Delta_{wr}$.

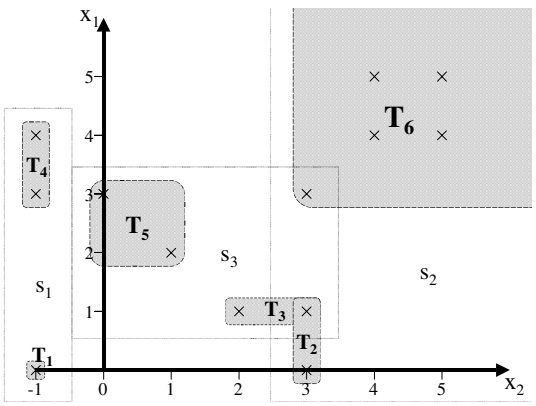


Figure 12. Distribution strategy based on tuple domains that are neither complete nor disjoint. The tuples and ι_F the same as in Figure 10. There are only three servers $\{s_1,s_2,s_3\}$. The servers' rectangles are implicitly given by the tuples that they store. The tuples are distributed to $\delta_\Sigma(T_1)=\{1\}=\delta_\Sigma(T_4)$, $\delta_\Sigma(T_2)=\{2\}=\delta_\Sigma(T_6)$ and $\delta_\Sigma(T_3)=\{3\}=\delta_\Sigma(T_5)$. Therefore, the servers' tuple domains are $\Sigma_1=[0,4]\times[-1,-1]$, $\Sigma_2=[0,\infty]\times[3,\infty]$ and $\Sigma_3=[1,3]\times[0,3]$.

In general, tuple domains are disjoint, if the hypercubes of the stored tuples are points. E.g. this condition holds if only actual tuples are stored and ι_F maps actual fields to one point intervals. Then, the two proposed definitions of tuple domains are alike with regard to disjointness.

If data is retrieved by a template T, every server q with $I_{\Im}(\iota_F)(T) \cap \Sigma_q \neq \varnothing$ has to be taken into account, regardless of $\iota_F \in I_F^{\subset}$ or $\iota_F \in I_F$. E.g. for $\iota_F \in I_F^{\subset}$, if $I_{\Im}(\iota_F)(t_1) = [0,2] \times [0,0]$ and $I_{\Im}(\iota_F)(t_2) = [0,2] \times [1,2]$ are stored on server q, then $\Sigma_q = [0,2] \times [0,2]$. If $I_{\Im}(\iota_F)(T) = [0,3] \times [0,1]$ for template T, it is $I_{\Im}(\iota_F)(t_1) \subset I_{\Im}(\iota_F)(T)$ but $\Sigma_q \not\subset I_{\Im}(\iota_F)(T)$. Therefore, server pruning is indifferent to $\iota_F \in I_F^{\subset}$ or $\iota_F \in I_F$. Figure 13 concludes in giving an overview of the different distribution strategies based on tuple domains.

		$\iota_F \in I_F$	$\iota_F \in I_F^{\subset}$
necessary condition for $\exists T \in \Im_n(q)$: $match_{\Im}(T^*,T)$	based on tuple domains	$\iota_{\Im}(T^*) \cap \Sigma_q \neq \varnothing$, i.e. $q \in \delta_\Sigma(T^*)$	
	based on the hypercubes of stored tuples	$\iota_{\Im}(T^*) \cap \iota_{\Im}(T) \neq \varnothing$	$\iota_{\Im}(T) \subseteq \iota_{\Im}(T^*)$
permissible distributions	disjoint tuple domains, i.e. $\forall T \in \Im_n(q)$: $\iota_{\Im}(T) \cap \Sigma_q \neq \varnothing$	δ_Σ	δ_Σ , ($\delta_{\Sigma,1}$, δ_Σ)
	overlapping tuple domains, i.e. $\forall T \in \Im_n(q)$: $\iota_{\Im}(T) \subseteq \Sigma_q$	δ_Σ , ($\delta_{\Sigma,1}$, δ_Σ)	

Figure 13. It is shown how $\iota_{\Im} = I_{\Im}(\iota_F)$ affects the distribution based on tuple domains. The first and second line give necessary conditions for pruning servers and tuples respectively. Letter condition can be exploited, if the set of hypercubes is accessible, i.e. on the server itself. The last two lines indicate that the distribution δ_Σ is mandatory, if $\iota_F \notin I_F^{\subset}$ and the tuple domains are disjoint. However, δ_Σ should be avoided, otherwise tuples are replicated among servers.

5.4 Formal Scalability Analysis of the Concept

In contrast to distributions based on hashing, the distributions introduced in this chapter are more complex. Hence, a formal analysis of their scalability based on the deterministic model of section 4.1 is difficult and is highly dependent on ι_F. E.g. for a constant ι_F, the tuple space does not scale. Alternatively, simulative methods or performance tests may be applied, in order to evaluate the scalability of the concept. This section is focused on analyzing the concept with an eye to the deterministic scalability model. The computation complexity of the distribution is not taken into account.

The finest granularity in the hypercube concept are points. If tuples are distributed by δ_Σ or δ_G, 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

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

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

Proof. If δ_Σ is applied, let q denote an arbitrary 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 distributions that are based on tuple domains. For δ_Σ, the implication given in Theorem 5.4.1 is proven to be an equivalence by Theorem 5.4.2.

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

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

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).

In general, A_w does not scale for δ_Σ. Hence, it is favourable to distribute with $(\delta_{\Sigma,1}, \delta_\Sigma)$, if it is permissible. It is clear that $S_Q(q)$, A_R and A_r suffer from overlapping tuple domains and templates with large hypercubes. However, an analysis requires an explicit definition of ι_F and of the algorithm that merges and splits tuple domains. Therefore, a formal analysis is relinquished.

5.5 Comparison to Approaches in Related Research Areas

Several approaches that aim at scalability have been developed in other research areas. They share in common the use of specific data structures, e.g. indices in relational databases. Hence, an abstract representation is assigned to data that is native to the respective data model. The mapping has to preserve important properties of the data. If a condition on such properties is given, it is possible to express sufficient or necessary conditions based on the abstract representation. For example in relational databases, the equality of indices is a necessary condition for equality of tuples. Favourable characteristics in computation and storage are mandatory for the data structures that are used as abstract representation. E.g. indexing takes advantage of hashing algorithms. In conclusion, the main challenge of defining the mapping ensues from finding an appropriate tradeoff between an appropriate abstract representation and its algorithmic characteristics. This applies also to the mapping that is proposed in this chapter. E.g. nested tuples are not directly supported, in order to take advantage of the experience in spatial data structures, i.e. hypercubes.

Yet another concept groups data according to its similarity or structure. Then, the data set is partitioned into distinct sets. For example, tuples are partitioned into relations in relational databases, objects are partitioned into extents in object oriented databases and semi structured data is partitioned into subsets according to their data scheme. Multiple tuples spaces [33] enable partitioning, too. However, the similarity of tuples in one space is purely semantic, so that partitions contain structurally heterogenous tuples. In general, the granularity of semantic partitioning is coarser than structural partitioning. In addition, contrarily to relations and extents, the use of multiple spaces is not enforced. Therefore, scalability concepts for tuple spaces should not depend on multiple spaces.

As for the placement of data, it is suggested that data is situated logically near to the participating entities that write or retrieve it. If the usage profile of entities is known in advance, data can be bound statically to appropriate servers. This is feasible for some application areas of databases, since such information may be added to scheme definitions. Otherwise, the database has to adapt dynamically to its environment. The application of this concept has been studied for tuple spaces in [50]. As it is shown in section 2.3, this approach aims at performance, but scalability is not achieved in general. Therefore, the concept of this chapter does not exploit logical proximity. Yet, adaptive placement of tuples is complementary and can be combined with the proposed concept.

As for data retrieval in database, query languages support operations like selection and projection, which reduces the size of the result set. E.g. the retrieval of data can be confined to a considerable subset of servers, if the selection predicate corresponds to an operation on the abstract representation. In tuple spaces, there is no notion of result sets. The structure of the retrieved data is induced by the template, so that it is impossible to define projections. However, templates enable selections, although the selection predicate is implicitly given by the template itself. Furthermore, current tuple space implementations have lost direct control of the selection predicate, since they allow customized matching. Hence, the selection predicate has to be evaluated on every tuple, in order to retrieve matching tuples from the tuple space. The concept of this chapter introduces a necessary condition for matching, i.e. the non-null intersection of hypercubes. The selection predicate is made explicit by the definition of the mapping of fields to intervals and by E=1. Therefore, selective data retrieval is confined to a subset of tuples.

Semi structured databases allow structural data retrieval, so that it is difficult to conceive scalable data retrieval. E.g. parts of the structure may be wildcarded. In order to process such queries, an indexing mechanism has been suggested in [42]. The indices are intervals and capture the ancestor-descendent relationship, which is similar to this chapter' s concept. Yet another approach [9] proposes a priori definitions of frequently used structural queries. The database administrator is in charge of finding and defining appropriate patterns. Both strategies have to compute and store metadata in advance, in order to increase performance. Despite the lack of schema definitions in tuple spaces, these strategies are useful for the definition of an appropriate mapping ι_F.

The necessity of user defined indexing and its application to object-relational databases is discussed in [16]. Object-relational databases allow user-defined predicates and data types. If the indexing of user-defined data types takes into

account frequently queried properties, retrieval of data may be contained to a considerable subset. The approach distinguishes equivalent and necessary conditions that are based on an abstract representation, i.e. the index. Therefore, selections can exploit the structure of user-defined types. In comparison to tuple spaces, fields and matching are user-defined too, so that the concept of this chapter is analogous. Besides, the only predicate is the test whether a tuple is matched by a template. However, an abstract representation of tuples is deduced automatically from the indices of its fields, i.e. intervals. Therefore, there exists a necessary condition for tuple matching. Furthermore, the definition of ι_F is facilitated.

5.6 Semantic and Nested Tuples

Semantic tuples. In order to retrieve tuples by structure, semantic tuples have been suggested in [31]. In general, structural expressiveness is restricted to nested tuples. According to the definition of nested tuples, $\perp_F$ matches every substructure, i.e. fields and trees of fields. Hence, semantic tuples can be substituted by nested tuples for structural queries.

Alternatively, matching of semantic templates may be evaluated by a disjunctive concatenation of matching predicates. E.g. a tuple is matched by the semantic template { `(int)`,`(string)` }, if the tuple is matched either by `(int)` or by `(string)`. Semantic tuples are not directly supported in current tuple space implementations. Hence, the following supposes that semantic tuples are implemented as list of tuples. Then, the complexity of creation, storage and matching of an arbitrary semantic tuple is proportional to its cardinality. However, the hypercube of a semantic tuple *T* can be determined independently of its cardinality, if the mapping $\iota_\Gamma: \Gamma \times Nat \rightarrow J(Nat)$ with $\iota_\Gamma(T, i) := \Pi_i(I_3(\iota_F)(T))$ takes O(1) for every dimension *i*. This is achieved by automatically evaluating ι_Γ while creating or updating the semantic tuple. Note, that $\iota_\Gamma(T, i) = \upsilon(\iota_F(\{\Pi_i(t) \mid t \in T\}))$, so that the complexity of creation and updating is not increased. In conclusion, this chapter's concept is valid for semantic tuples, if they are supported by the tuple space implementation (C=0).

Nested tuples. This chapter's concept can be extended by defining an appropriate abstract representation of nested tuples. Based on tuples *t* with $\text{depth}(t) = 1$ that are mapped to hypercubes, the abstraction is defined inductively. E.g. a tuple *t* with $\text{depth}(t) = 2$ is mapped to a vector of hypercubes, etc. Therefore, the abstract representation of nested tuples is complex and possibly hampers the concept's efficiency.

However, there are only few implementations that directly support nested tuples. Theorem 3.3.6 suggests the simulation of nested tuples by tuples that are vectors of fields. In most cases, such a simulation is applicable and the tuple space implementation does not have to support nested tuples. In addition, this approach avoids complex abstract representations of nested tuples.

In contrast, it may be attempted to map every node of a nested tuple's tree to an additional tuple, as it is true for mapping semi structured data to relational databases. Then, the atomicity and isolation of tuple storage and retrieval is not

guaranteed any more. This ensues from the lack of appropriate locking and scheduling mechanisms in tuple spaces.

In conclusion, this chapter' s concept assumes that nested tuples are either disallowed or simulated. If this proves to be too restrictive in any application area of tuple spaces, the set of fields has to be enlarged, in order to include semi structured data. For example, an XML [10] class can be defined and be inherited by several subclasses that capture different data schemes. Note, that in general B=0, if instances of semi structured data are modeled as actual fields. Therefore, the definition of an appropriate ι_F becomes difficult. Nevertheless, it is possible to take advantage of approaches to index semi structured data, that are introduced in section 5.5.

5.7 Spatial Data Structures

The concept of this chapter introduces an abstract representation of tuples and tuple domains, i.e. hypercubes. Hence, the elements of the underlying data structure are spatially extended objects. Consequently, the spatial data structure has to support dynamic insertion, deletion, intersection and inclusion queries. Such data structures have been studied in the past. An overview is given in [30, 51]. They generalize data structures that manage one dimensional points, like the B-tree [4]. Alternatively, hypercubes are transformed to a representation, that avoids multiple dimensions or spatially extended objects, as it is true for space ordering and hyperpoints respectively.

This section is focused on data structures that suit particularly well to the hypercube concept. Such a data structure is not only applied for server pruning, but it is also beneficial for tuple pruning on an arbitrary server. Components that prune servers, are assumed to keep the hypercubes of the tuple domains in main memory. This assumption is reasonable, since a tuple domain is defined by 2·d integers. Then, the storage complexity of tuple domains is negligible. Furthermore, the pruning of servers or tuples should be effective, i.e. a maximum of servers or tuples has to be pruned. However, the computation complexity of pruning should be low.

As for the storage of tuples, the servers have to hold about the same number of tuples. Even for $\iota_F \notin I_F^{\subset}$, the spatial data structure should not require that a tuple is stored on several servers. In addition, the number of servers should swiftly adjust to the number of tuples, so that p is gradually incremented or decremented. Furthermore, splitting or merging tuple domains is performed locally, i.e. by the respective servers, so that the performance of the other servers is not affected.

The data structure has to cope with non uniform distributions of hypercubes in the hyperspace. This is particularly valid, if the dimension of most tuples is considerably lower than d. In addition, the data structure should not become inefficient, if tuples with large hypercubes are stored.

Hyperpoints. It has been suggested to take advantage of spatial data structures that is specialized on points. Therefore, d-dimensional hypercubes are mapped to 2·d-dimensional hyperpoints in the dual space. Queries that test the intersection or inclusion of hypercubes, are transformed to equivalent queries in the dual space.

E.g. for d=1, a tuple *t* with $I_3(\iota_F)(t) = [2,5]$ is mapped to $[2,2] \times [5,5]$. For template *T* with $I_3(\iota_F)(T) = [a,b]$, the intersection query is transformed to the dual space by querying hyperpoints that are element of $[-\infty,b] \times [a,\infty]$.

As a result, the spatial data structure has to enable unbounded range queries. Furthermore, the hyperpoints are not uniformly distributed in the dual space. In addition, the transformation to hyperpoints does not preserve proximity of hypercubes. In general, the hypercubes of stored tuples are small or points, so that the approach of transforming to hyperpoints does not suit well. Finally, the servers' tuple domains are transformed to hyperpoints as well. Hence, the tuple domains do not partition the hyperspace any more.

Additive Methods. Yet another approach tests intersection or inclusion queries independently for each dimension. The result of the query is computed by intersecting the result sets of every dimension. Therefore, a *d*-dimensional query is reduced to *d* one-dimensional queries. Then, *d* interval data structures, like range trees [6], are managed simultaneously. E.g. this approach is applied in databases that compute several indices per data entry. Since the total complexity of a query is the sum of the complexity of its subqueries, the approach is called additive. It is particularly efficient, if the dimension of the templates is considerably lower than *d*. Furthermore, queries can be computed in parallel.

However, each one-dimensional query is performed independently, so that it cannot make use of pruning in other dimensions. E.g. for d=2, if no server with odd index is in the result set of the first dimension, the query of the second dimension still looks for odd servers and includes some of them in the result set. This deficiency becomes even more apparent for higher dimensions. In addition, the total size of all one-dimensional result sets is $\Omega(d)$ times larger than the result set of the original query, as it is true for the computation complexity of the intersection. Furthermore, the definition of a server' s tuple domain is distributed to *d* data structures. Hence, its alteration takes $\Omega(d)$, and information about tuple domains, like adjacency, completeness and disjointness, is spread.

Space Ordering. Another approach exploits the countability of the hyperspace, in order to transform it to the linear space. The transformation has to be fast and invertible. In addition, the proximity of objects should be preserved by the mapping. *Morton orders* [49] suit best to these requirements, even though the preservation of proximity is not guaranteed. Figure 14*(a)* illustrates Morton orders, that are also known as *N orders* or *Z orders*. The transformation $\mu: Nat \times (Nat \cup \{-1\})^{d-1} \rightarrow Nat$ interleaves the binary representation of each dimension, i.e.

$$\mu(P) := p_j^{(d)} \ldots p_j^{(1)} \ldots p_1^{(d)} \ldots p_1^{(1)} p_0^{(d)} \ldots p_0^{(1)}$$

with $p_j^{(i)} \ldots p_1^{(i)} p_0^{(i)}$ being the binary representation of $\Pi_i(P) + 1$ for $i > 0$ and of $\Pi_0(P)$ for $i = 0$. An arbitrary hypercube *C* is specified by two hyperpoints lb(C) and ub(C), that are the lower and upper bounds in every dimension respectively, i.e.

$$lb(C) := (\min(\Pi_1(C)), \ldots, \min(\Pi_d(C))) \,, \quad ub(C) := (\max(\Pi_1(C)), \ldots, \max(\Pi_d(C))) \,.$$

Then, $\mu_C: J(Nat) \times J(Nat \cup \{-1\})^{d-1} \rightarrow J(Nat)$ maps hypercubes to intervals, as it is defined by

$$\mu_C(C) := [\ \mu(\ lb(C)\ ),\ \mu(\ ub(C)\ )\ ]\ .$$

The interval $\mu_C(C)$ implicitly defines the set $\mu_C^*(\mu_C(C))$ of hyperpoints P that comply with $\mu(P) \in \mu_C(C)$. This is illustrated in Figure 14*(b)*. The transformation preserves necessary conditions for tuple matching, as it is proven in following theorem. Therefore, server and tuple pruning may be performed on intervals instead of hypercubes.

Theorem 5.7.1. Let $C,C' \in J(Nat) \times J(Nat \cup \{-1\})^{d-1}$ and $P,P' \in Nat \times (Nat \cup \{-1\})^{d-1}$. In addition, let $\leq_D$ denote the dominance relation on an arbitrary hyperspace, i.e. $P \leq_D P'$ if and only if $\forall i \in \{1,\ldots,d\}: \Pi_i(P) \leq \Pi_i(P')$. Then, it is

a) $P \leq_D P' \rightarrow \mu(P) \leq \mu(P')$
b) $C \subseteq \mu_C^*(\mu_C(C))$
c) $C \cap C' \neq \emptyset \rightarrow \mu_C(C) \cap \mu_C(C') \neq \emptyset$
d) $C \subseteq C' \rightarrow \mu_C(C) \subseteq \mu_C(C')$.

Proof.

a) Let k denote the highest index, so that the vectors $(p_k^{(d)},\ldots,p_k^{(1)})$ and $(p'_k{}^{(d)},\ldots,p'_k{}^{(1)})$ differ. Since P is dominated by P', it is $p_k^{(i)} \leq p'_k{}^{(i)}$ for every $i \in \{1,\ldots,d\}$. Hence, $\mu(P) \leq \mu(P')$ ensues from the definition.
b) Let P denote a hyperpoint in C. Hence, $lb(C) \leq_D P \leq_D ub(C)$. Then, $\mu(lb(C)) \leq \mu(P) \leq \mu(ub(C))$ ensues from a). Therefore, $\mu(P) \in \mu_C(C)$.
c) Let $P \in C \cap C'$, then $\mu(P) \in \mu_C(C) \cap \mu_C(C')$ ensues from b).
d) $C \subseteq C'$ implies $lb(C') \leq_D lb(C) \leq_D ub(C) \leq_D ub(C')$. Hence $\mu(lb(C')) \leq \mu(lb(C))$ and $\mu(ub(C)) \leq \mu(ub(C'))$ ensue from a).

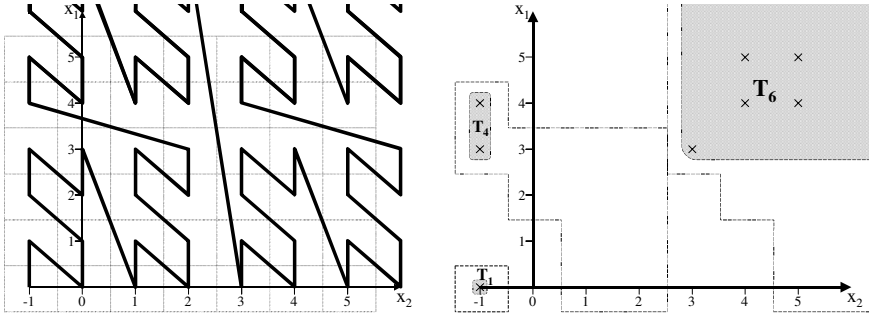


Figure 14. On the left *(a)*, the hyperspace is numbered by the Morton order. On the right *(b)*, the transformation of tuples' hypercubes C to intervals is indicated by drawing the boundaries of $\mu_C^*(\mu_C(C))$. The tuples are the same as in Figure 10.

However, the implications of Theorem 5.7.1 cannot be replaced by equivalences, as it is clear from Figure 14*(b)*. Even if tuple domains are disjoint in the hyperspace, they generally overlap after their transformation. Therefore, it seems promising to define tuple domains by intervals [3]. Then, a tuple domain Σ_q is described by only two integers, but $\mu_C^*(\Sigma_q)$ is not a hypercube any more. If each

tuple domain is a superset of the stored tuples' hypercubes, the overlapping area of tuple domains becomes considerable. E.g. in Figure 14*(b)*, $\mu(I_3(\iota_F)(T_4))$ is six times larger than $I_3(\iota_F)(T_4)$. Therefore, $\iota_F \in I_F^{\subset}$ has to be assumed, in order to apply the distribution $(\delta_{\Sigma,1}, \delta_\Sigma)$.

Space ordering is bijective, so that integer indices of the linear space are *d* times longer than original integer indices of an arbitrary dimension. Therefore, the description of tuple domains and tuples' intervals still takes $\Theta(d)$ in complexity. In addition, the definition of a necessary condition for tuple matching is further watered-down, so that the effectiveness of pruning deteriorates. Besides, the interleaving of dimensions is fixed by the spatial order. Hence, tuple domains cannot adapt to usage profiles that are irregular with regard to the dimension of stored tuples. E.g. the dimension of tuples is frequently considerably lower than *d*. In Figure 14*(b)*, $\mu_C^*(\mu(I_3(\iota_F)(T_4)))$ covers two dimensions, even though the dimension of T_4 is one.

In order to overcome some of these deficiencies, the space ordering approach is refined in [3]. The size of an integer is reduced by storing a prefix and assuming a default value for the remaining bits. However, this is only efficient, if certain integers are favoured for the description of tuples and tuple domains. Furthermore, it is suggested that the necessary condition is evaluated based on $\mu_C^*(\Sigma_q)$ in the hyperspace. Yet this implies that queries do not exploit space ordering.

Nevertheless, the space ordering approach may be applied for nested tuples and high dimensional tuples. It enables iterative indexing of nested tuples, which provides the foundation of defining an appropriate ι_F. In addition, the dimension of tuples can be arbitrarily decreased, so that a fixed maximal dimension *d* imposes no restriction on the tuple space.

Overlapping Partitions. Upper approaches transform hypercubes to a more manageable representation. Alternatively, there a spatial data structures that partition the hyperspace, which is analogous to partitioning the set of tuples into tuple domains. Such an approach suits well to this chapter's concept, because tuple domains are directly represented as partitions in the data structure. If the distribution $(\delta_{\Sigma,1}, \delta_\Sigma)$ is applied for $\iota_F \notin I_F^{\subset}$, tuple domains have to be supersets of the tuples that they store. Then, tuple domains are not disjoint, as it is shown in Figure 12. Therefore, the spatial data structure has to manage overlapping partitions.

The R-tree [37] complies with this demand. The leaves of the tree are hypercubes of the stored tuples. Then, the hypercube *C* of an arbitrary node is recursively defined by

$$C := \upsilon\left(\bigcup_{i=1}^{b} C_i\right),$$

with $C_1,\ldots,C_b$ depicting the hypercubes of the sons. As a result, the set of tuple domains is an arbitrary anti-chain of nodes, such that every leaf is the descendent of exactly one tuple domain. Note, that the root's hypercube is a subset of the hyperspace. In general, the tuple domains are neither complete nor disjoint. Figure 15 illustrates such a tree.

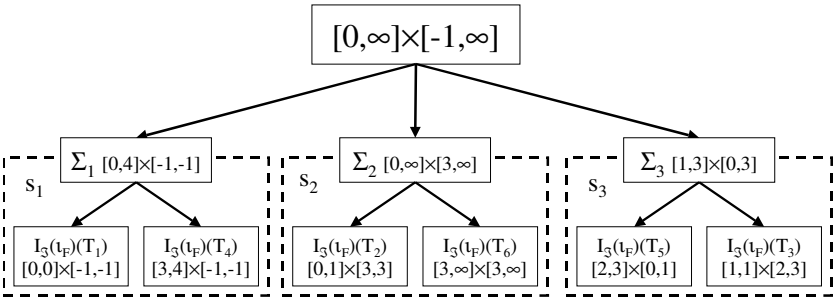


Figure 15. For the tuple domains of Figure 12, the respective R-tree is illustrated. The dashed areas indicate parts of the tree, that are stored and managed locally. Note, that only s_1 is pruned for the template T=(address,address) with $I_3(t_F)(T) = [3,3]\times[3,3]$, even though obviously s_3 should be pruned, too.

This approach excels by employing a single data structure for server and tuple pruning. Every server manages the subtree of its tuple domain. A subtree is kept local and enables tuple pruning on the respective server. The remaining part of the hypercube tree is globally accessible. Its leaves are the respective nodes of the tuple domains, so that it provides the foundation of server pruning. As a result, merging and splitting of tuple domains is accomplished by reassigning tuple domains to nodes of the hypercube tree.

However, pruning may not be effective, if the overall overlapping area is large. E.g. even if the hypercube of a template is a hyperpoint, the query is generally performed on more than one server. Therefore, pruning effectiveness deteriorates, if tuples with large hypercubes have to be stored in the tuple space. Furthermore, the insertion of tuples is ambiguous, if their hypercubes intersects with several tuple domains. Hence, some heuristic has to identify the tuple domain that is optimal with regard to the overall overlapping area or with regard to the size of the tuple domains. Besides, the alteration of a node' s hypercube may affect the hypercubes of its ancestral nodes. Therefore, tuple insertion and deletion has to be propagated up the tree.

If the hypercube tree is balanced, server pruning takes O(log p). However, it is difficult to undo splitting of nodes that are close to the root. As a result, the cohesion of the sons' hypercubes may deteriorate. Therefore, the hypercube tree has to be restructured periodically. E.g. the R^*-tree [5] refines the splitting strategy of the R-tree. If a tuple domain is splitted, some tuples of the respective server are reinserted into the tuple space. As a result, splitting the domain of a server cannot be performed locally any more.

Disjoint Partitions. Yet another approach prohibits overlapping tuple domains. Then, it is guaranteed that hyperpoint queries are directed to only one server, so that server pruning is generally more effective.

The R^+-tree [55] applies this idea to the R-tree. Apart from the leaves, the hypercubes of an arbitrary anti-chain of nodes are disjoint. However, the hypercube of the stored tuples may still overlap. Therefore, a leaf' s hypercube is not required to be a subset of its father' s hypercube any more. It is only enforced

that the hypercubes of a leaf and its father are not disjoint. As a result, the notion of tuple domains is water-down, so that the traversal of the tree during queries is hardly optimal. Nevertheless, the union of tuple domains has to be a superset of a stored tuple' s hypercube. Therefore, the insertion of a tuple may necessitate the enlargement of an ancestral hypercube, if the tuple domains are not complete. In such a case, it has been proven [30] that the disjointness restraint may cause a deadlock. Furthermore, the distribution ($\delta_{\Sigma,1}$, δ_{Σ}) is not permissible for $\iota_F \notin I_F^{\subset}$, as it is shown in Figure 13. However, the use of δ_{Σ} implies that tuples are replicated among servers, so that insertions and deletions cannot be performed locally. In addition, tuple domains are splitted more frequently, since the servers store redundant information. Splitting may not effective in decreasing the server load, if the majority of the domain' s tuples are replicated. In conclusion, the following assumes $\iota_F \in I_F^{\subset}$ and complete tuple domains. Figure 16 illustrates such a R^+-tree.

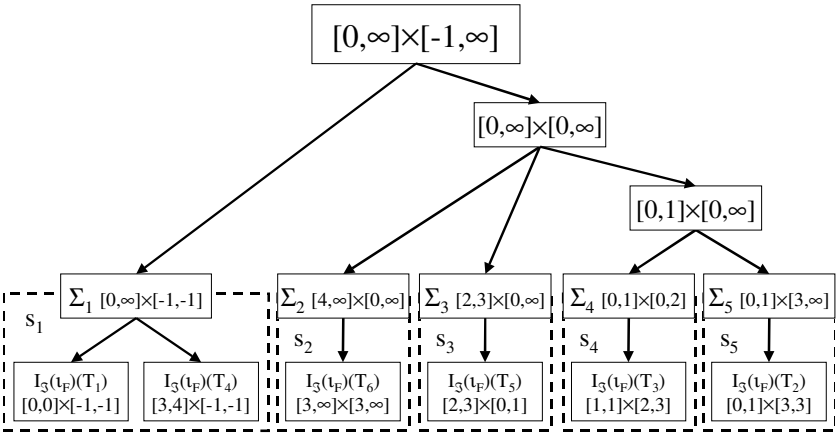


Figure 16. For the complete tuple domains of Figure 11, the respective R^+-tree is illustrated. The dashed areas indicate parts of the tree, that are stored and managed locally. It is assumed that the tuples T_3 and T_6 are stored on s_4 and s_2 respectively, even though it is possible to store them on s_5 and s_3 respectively.

Alternatively, the extended k-d-tree [44] systematically decomposes the hyperspace into disjoint and complete hypercubes. A hypercube is splitted by an iso-oriented hyperplane, so that every non leaf node has exactly two sons. However, this implies that a node can only be merged with its buddy. Therefore, the heuristic for hypercube splitting is simplified at the expense of the applicability of hypercube merging. Nevertheless, splitting and merging is performed locally. Figure 17 gives an example for extended k-d-trees.

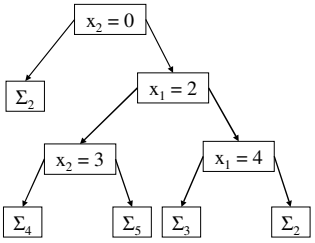


Figure 17. For the tuple domains of Figure 11, an excerpt of the respective extended k-d-tree is illustrated. The local parts of the tree are omitted in the figure. The hyperspace is decomposed by iso-oriented hyperplanes. E.g. the left and right son of the root node store hypercubes with $x_1<0$ and $x_1\geq 0$ respectively.

There are several other data structures that partition the hyperspace into disjoint partitions. However, they are not suited for this chapter's concept. E.g. priority search trees [21] are designed for hyperspaces with few dimensions. Quadtrees [28] prohibit gradual adjustment of the number of servers.

Conclusion. Figure 18 summarizes this section's analysis of spatial data structures and their fitness to this chapter's concept. In general, the R-tree suits best, among others because of its simplicity. For $\iota_F \in I_F^{\subset}$, the R-tree is outperformed by the R^+-tree and the extended k-d-tree. Even though being inferior to partitioning, space ordering generalizes the hypercube concept with regard to nested and high dimensional tuples.

Requirement		Hyper-points	Additive Methods	Space Ordering	Hyperspace Partitions			
					overlapping		disjoint	
					R	R^*	R^+	E.k-d
Suitability for both, server and tuple pruning		-	-	+	++		+	
Tuple domains may be kept in main memory		?	?	+/?	+		+	
Pruning	complexity	-	--	+	o	+	o	+
	effectiveness	o	o	-	-	o	o	
Storage	balanced	-/?	?	?	+		+	o
	no forced replication	?	?	-/?	+		+	
	large cubes	?	?	-/?	-		+	
Adjustment of *p*	gradual	?	o	+	+		+	
	local	?	o	+	+	-	+	
Adaptivity to non uniform distribution	in general	-/?	o	o	+		+	
	for tuples with low dimensions	?	+	--	+		+	

Figure 18. Important properties of spacial data structures are listed in the eye of their suitability to this chapter' s concept. The question mark indicates that the suitability of the respective property depends on the employed data structure. E.g., the hyperpoint approach is not restricted to a specific point access method. For the disjoint partitioning approach, it is assumed that $\iota_F \in I_F^{\subset}$. Otherwise it is not competitive.

Appendix A: References

1. Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wienner, J.: The Lorel Query Language For Semi-Structured Data. International Journal of Digital Libraries, page 68-88, Volume 1 (1997)
2. Anderson, B., Shasha, D.: Persistent Linda: Linda + Transactions + Query Processing. In J.P. Banatre and D. Le Metayer, editors, Research Directions in High-Level Parallel Programming Languages, volume 574 of Lecture Notes in Computer Science. Springer Verlag (1991)
3. Bayer, R.: The Universal B-Tree for Multidimensional Indexing. Internal Report, Technische Universität München (1996)
4. Bayer, R., McCreight, E.: Organization and maintenance of large ordered indices. Acta Informatica 1, 3, 173-189 (1977)
5. Beckmann, N., Kriegel, N.-P., Schneider, R., Seeger, B.: The R^*-tree: An Efficient and Robust Access Method for Points and Rectangles. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 322-331 (1990)
6. Bentley, J. L.: Decomposable Searching Problems. Information Processing Letters 8, 5, 244-251 (1979)
7. Berners-Lee, T.: Universal Resource Identifiers in WWW. Internet informational RFC1630 (1994)
8. Bjornson, R. D.: Linda on Distributed Memory Multiprocessors. PhD thesis, Yale University, TR931 (1993)

9. Bjornson, R. D., Carriero, N., Gelernter, D.: From Weaving Threads to Untangling the web: A View of Coordination from Linda' s Perspective. Multiple tuple spaces in Linda. In Coordination ' 97, Lecture Notes in Computer Science, pages 1-17. Springer-Verlag (1997)
10. Bray, T., Paoli, J., Sperberg-McQueen, C. M.: Extensible Markup Language (XML). http://www.w3.org/TR/PR-xml.html (1997)
11. Busi, N., Gorrieri, R., Zavattaro, G.: A process algebraic view of Linda coordination primitives. Theoretical Computer Science, 192(2):167-199 (1998)
12. Campbell, D., Osborne, H., Wood, A.: Characterising the Design Space for Linda Semantics. Technical Report YCS-97-277, University of York (1997)
13. Carriero, N., Gelernter, D., Zuck, L.: Bauhaus Linda. In Paolo Ciancarini, Oscar Nierstrasz and Akinori Yonezawa, editors. Object-Based Models and Languages for Concurrent Systems, volume 924 of Lecture Notes in Computer Science, pages 66-76, Springer-Verlag (1995)
14. 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)
15. Cellary, W., Gelenbe, E., Morzy, T.: Concurrency Control in Distributed Database Systems. North-Holland (1988)
16. Chen, W., Chow, J., Fuh, Y., Grandbois, J., Jou, M., Mattos, N. M., Tran, B., Wang, Y.: High Level Indexing of User-Defined Types. Proceedings of the 25th International Conference on Very Large Databases (VLDB 1999), pages 554-564, Edinburgh, UK (1999)
17. Cohn, P., M.: Algebra, John Wiley & Sons, Second Edition (1982)
18. Codd, E. F.: A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, 13:6, 377-387 (1970)
19. Cooper, B. F., Sample, N., Franklin, M. J., Hjaltason, G. R., Shadmon, M.: A Fast Index for Semistructured Data. 27th International Conference on Very Large Data Bases (VLDB 2001), Rome, Italy (2001)
20. 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)
21. McCreight, E. M.: Priority Search Trees, SIAM J. Computing 14, Pages 257-276 (1985)
22. Date, C. J., Darwen, H.: A Guide to the SQL Standard. Third edition, Addison-Wesley (1993)
23. Davies, N., Wade, S. P., Friday, A., Blair, G. S.: Limbo: A Tuple Space Based Platform for Adaptive Mobile Applications. Proceedings of the International Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP ' 97), Toronto, Canada, pp291-302 (1997)
24. Devlin, B., Gray, J., Laining, B., Spix, G.: Scalability Terminology: Farms, Clones, Partitions, and Packs: RACS and RAPS. Microsoft Research, Technical Report MS-TR-99-85 (1999)
25. ECOM (Ed.): Electronic Commerce – An Introduction. http://ecom.fov.uni–mb.si (1998)
26. Dictionary of Computing. http://wombat.doc.ic.ac.uk (2000)
27. Feller, W.: Introduction to Probability Theory and Its Applications. Volume II. Wiley Series in Probability and Mathematical statistics (1970)
28. Finkel, R., Bentley, J. L.: Quad Trees: A Data Structure for Retrieval of Composite Keys. Acta Informatica 4(1), 1-9 (1974)
29. Franklin, S., Graesser, A.: Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents. Proceedings of third Int. Workshop on Agent Theories, Architectures and Languages (1996)

30. Gaede, V., Gunther, G.: Multidimensional Access Methods. ACM Computing Surveys (1997)
31. 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)
32. Gelernter, D.: Generative communication in Linda. ACM Transactions on Programming Languages and Systems, 7(1): 80-112 (1985)
33. 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)
34. Gelernter, D., Zuck, L: On What Linda Is: Formal Description of Linda As a Reactive System. In Coordination ' 97, Lecture Notes in Computer Science, pages 187-204. Springer-Verlag (1997)
35. Goldman, R., Chawathe, S., Crespo, A., McHugh, J. A.: Standard Textual Interchange Format for the Object Exchange Model (OEM). Department of Computer Science, Stanford University. California, USA (1996)
36. V. d. Goot, R., Schaeffer, J., Wilson, G. V.: Safer Tuple Spaces. In Coordination ' 97, Lecture Notes in Computer Science, pages 289-301. Springer-Verlag (1997)
37. Guttman, A: R-trees: A dynamic index structure for spatial searching. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 47-57 (1984)
38. Van Huizen: JMS, an Infrastructure for XML-based Business-to-Business Communication, JavaWorld (2000)
39. IBM Systems: Enterprise TSpaces. http://www.almaden.ibm.com/cs/TSpaces (2001)
40. 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)
41. Lausen, G., Vossen, G.: Models and Languages of Object-Oriented Databases. Addison-Wesley (1997)
42. Li, Q., Moon, B.: Indexing and Querying XML Data for Regular Path Expressions. Proceedings of the 27th International Conference on Very Large Databases (VLDB 2001), pages 361-370, Rome, Italy (2001)
43. 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)
44. Matsuyama, T., Hao, L. V., Nagao, M.: A File Organization for Geographic Information Systems Based on Spatial Proximity. Int. J. Comp. Vision, Graphics and Image Processing 26(3), 303-318 (1984)
45. De Nicola, R., Ferrari, G.-L., Pugliese, R.: Locality based Linda: programming with ex-plicit localities. Proceedings TAPSOFT ' 97. Lecture Notes in Computer Science 1214, 712-726, Springer Verlag (1997)
46. Obreiter, P.: Extending Tuple Spaces Towards a Middleware for eCommerce. Studienarbeit, University of Karlsruhe, Germany (2000)
47. Obreiter, P., Graef, G.: Applying Component Based Web Engineering in an International Enterprise. Proceedings of the International Forum cum Conference on Information Technology and Communication at the Dawn of the New Millennium, pages 33-45, Bangkok, Thailand (2000)
48. Obreiter, P., Graef, G.: Towards Scalability in Tuple Spaces. ACM Symposium of Applied Computing (SAC) Special Track on Coordination Models, Languages and Applications, Madrid, Spain (2002)

49. Peano, G.: Sur une courbe qui remplit toute une aire plaine. Mathematische Annalen 36, 157-160 (1890)
50. Rowstron, A.: WCL, a Coordination Language for Geographically Distributed Agents. World Wide Web Journal, Volume 1, Issue 3, 167-179 (1998)
51. Samet, H.: The Design and Analysis of Spatial Data Structures. Addison-Wesley, Reading MA (1990)
52. Schek, H.-J., Scholl, M. H.: The NF2 relational algebra for uniform manipulation of external, conceptual, and internal data structures. In J.W. Schmidt, editor, Sprachen fuer Datenbanken, IFB 72. Springer Verlag (1983)
53. Schoenfeldinger, W.: WWW Meets Linda. In Proc. 4th Int. World Wide Web Conference: The Web revolution, Boston, MA, World Wide Web Journal 1(1):259-276 (1995)
54. Schroeder, T., Goddard, S., Ramamurthy, B.: Scalable Web server clustering technologies. IEEE Network, May-June 2000, pp. 38-45 (2000)
55. Sellis, T., Roussopoulos, N., Faloutsos, C.: The R^+-tree: A Dynamic Index for Multi-dimensional Objects. In Proc. 13th Int. Conf. on Very Large Data Bases (VLDB), 507-518 (1987)
56. Stonebraker, M.: Object-Relational DBMSs - the Great Next Wave. Morgan-Kaufmann (1996)
57. Sun Microsystems: JavaSpaces Technology, http://java.sun.com/products/javaspaces (2000)
58. Sun Microsystems: Java, http://java.sun.com (2000)
59. Weiser, M.: Some Computer Science Issues in Ubiquitous Computing, Communications of the ACM (1993)
60. Wells, G., Chalmers, A.: An Extended Linda System Using PVM. In PVM Users' Group Meeting, Pittsburgh (1995)
61. Wyckoff, P., McLaughry, S. W., Lehman, T. J., Ford, D. A.: TSpaces, IBM Systems Journal (1998)