Semantic Tuplespace

Liangzhao Zeng[1], Hui Lei[1], and Badrish Chandramouli[2]

[1] IBM T.J. Watson Research Center, Yorktown Heights, NY 10598
{lzeng, hlei}@us.ibm.com
[2] Duke University, Durham, North Carolina 27708-0129
badrish@cs.duke.edu

Abstract. The tuplespace system is a popular cooperative communication paradigm in service-oriented computing. Tuple matching in existing tuplespace systems is either type-based or object-based. It requires that both tuple writers and readers adhere to the same approach of information organization (i.e., same terminologies or class hierarchy). Further, it examines the value of the tuple contents only. As such, these tuplespace systems are inadequate for supporting communication among services in heterogeneous and dynamic environments, because services are forced to adopt the same approach to organizing the information exchanged. In order to overcome these limitations and constraints, we propose a semantic tuplespace system. Our system uses ontologies to understand the semantics of tuple contents, and correlates tuples using relational operators as part of tuple matching. Therefore, by engineering ontologies, our system allows different services to exchange information in their native formats. We argue that a semantic tuplespace system like ours enables flexible and on-demand communication among services.

1 Introduction

The tuplespace paradigm is a simple, easy to use, and efficient approach for supporting cooperative communication among distributed services. Typically, a tuplespace system contains three roles: (i) tuple writers, who write tuples into sharespace, (ii) tuple readers, who read/take tuples that they are interested in, by specifying templates, and (iii) the tuplespace server, who is responsible for managing the sharespace and routing the tuples from writers to readers. The earliest tuplespace systems were *type-based*. A tuple in Linda [4] is a series of typed fields. For example, a tuple can be $t($`'Sports Car', 400,000`$)$. Tuple matching is based on a template that consists of a series of typed fields or type definitions. For instance, a template can be $\varphi(\langle$`'Sports Car'`$\rangle, \langle$`?float`$\rangle)$, where typed field (e.g., $\langle$`'Sports Car'`$\rangle$) requires value identical matching (e.g., string that is the same as `'Sports Car'`); while the type definition (e.g., $\langle$`?float`$\rangle$) only concerns the type matching (e.g., any float value). Obviously, such systems have limitations on specifying filtering criteria: either exact value or type matching. For above example, any tuples with type float in the second field can satisfy the template's requirement on second field, regardless of the value of the field.

Consequently, as an improvement to type-based solutions, *object-based* tuplespace systems have been proposed [10]. Instead of exact type matching, these systems

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 366–381, 2005.
© Springer-Verlag Berlin Heidelberg 2005

enable *object compatibility* based type matching. Further, these systems allow tuple readers to specify queries on fields, which provides the flexibility of choosing filtering criteria along multiple dimensions. For example, the template in the vehicle dealer example may be refined as φ'(⟨SportsCar⟩, ⟨CarInsurance, CarInsurance.premium < 2000⟩). This template indicates that those tuples that first field's type is SportsCar or descendent of SportsCar (e.g., USSportsCar, if USSportsCar is a descendent class of SportsCar in the implementation of the class hierarchy) and the second field's type is CarInsurance or descendent of CarInsurance and the premium is less than 2000, will be delivered to the reader.

Considering the adaptability and flexibility requirements from services that operate in dynamic environments, we argue that both type-based and object-based tuplespace systems are not sufficient in two aspects:

- Value-based matching. Currently, in object-based tuplespace systems, the type matching is based on object compatibility, wherein the relationship among the objects is deduced from the implementation of the class hierarchy. Without semantic support to understand the meaning of the field, the matching algorithm assumes that both tuple writers and readers share the same implementation of class hierarchy. Such an assumption is hard to enforce when the relationship of tuple writers and readers is dynamically formed.
- One-to-one matching. Presumably, services read multiple tuples in a transaction as no single tuple can provide all the necessary fields, when they interact with a collection of partner services. However, in current tuplespace systems, correlation of interrelated tuples is not supported, which requires custom implementation by application programmers. The implementation of tuple correlation is often a challenging and involving task. Further, it requires that the application programmers be aware of all the tuples that are provided by partner services in advance at development time. Such a requirement is impractical when a service has a dynamic collection of partners.

In this paper, we introduce a semantic tuplespace system. Our system enables semantic tuple matching, wherein semantic knowledge is maintained in ontologies. This releases the constraints in object-based tuplespace systems that writers, readers and the server must share the same implementation of class hierarchy. Unlike traditional tuplespace systems, tuple correlation in our system is performed by the tuplespace server, which is transparent to tuple readers. Therefore, services in dynamic environments become easier to develop and maintain as tuple semantic transformation and correlation can be provided as part of the tuplespace system. In a nutshell, the salient features and contributions of our system are:

1. Efficient semantic tuple matching. A naive approach to enabling semantic tuple matching is term generation, in which more generic fields (i.e., objects) are generated based on ontologies. For example, from an object of sportsCar, the system can generate a more generic object about car. Such an approach is clearly very inefficient, since it generates unnecessary redundant tuples. In our framework, instead of adopting term generation approach, the system enables semantic tuple routing by rewriting templates, wherein no redundant tuples need to be generated.

2. Semantic-based, correlation matching. With ontology support, it is possible for the system to conduct tuple correlation based on tuple content semantics using relational operators. For example, two tuples in a sharespace can be correlated to one by the join operator and then delivered to tuple readers. We extend tuple matching in traditional tuplespace systems with two kinds of correlation matchings, namely those based on common fields across tuples and those based on attribute dependence. Correlation matching can automatically search available tuples which can only provide partial information required by a read/take template, and correlate them to one tuple that contains all the fields required by the template.

The remainder of this paper is organized as follows: Section 2 introduces some important concepts and presents the overview of the semantic tuplespace system. Section 3 and 4 discuss two main features of the proposed system. Section 5 illustrates some aspects of the implementation. Finally, Section 6 discusses some related work and Section 7 provides concluding remarks.

2 Preliminaries

In this section, we first introduce some important concepts in ontology, and then present the proposed system architecture of the semantic tuplespace system. Finally, we outline the tuple matching algorithm.

2.1 Ontology

In our system, we adopt an object-oriented approach to the definition of ontology, in which the type is defined in terms of *classes*[1] and an instance of a class is considered as an *object*. In the subsection, we present a formal description of class and object. It should be noted that this ontology formulation can be easily implemented using OWL [11] framework. We will present details on how to use ontology to perform semantic matching and correlation matching in following sections.

Definition 1 **(*Class*).** A class C is defined as the tuple $C = \langle N, S, P, R, F \rangle$, where

- N is the name of the class;
- S is a set of synonyms for the name of class, $S = \{s_1, s_2, ..., s_n\}$;
- P is a set of properties, $P = \{p_1, p_2, ..., p_n\}$. For $p_i \in P$, p_i is a 2-tuple in form of $\langle T, N_p \rangle$, where T is a basic type such as integer, or a class in an ontology, N_p is the property name.
- R is a set of parent classes, $R = \{C_1, C_2, ..., C_k\}$;
- F is a set of dependence functions for the properties, $F = \{f_1, f_2, ..., f_l\}$. Each function is in form of $f_j(p_1^{'}, p_2^{'}, ..., p_m^{'})$ and associated with a predicate c, where the output of f_j is a property p_i of class C and $p_i^{'}$ is property from a class other than C and the predicate c is used to correlate $p_i^{'}$. □

[1] Some notations used in this paper are summarized in Table 1.

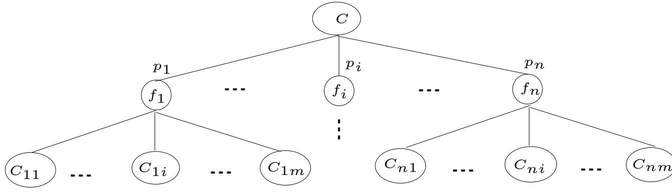

Fig. 1. A Dependence Tree of the Class C

In the definition of class, the *name*, *synonyms*, and *properties* present the connotation of a class; while *parent classes* and *dependence functions* specify relationships among the classes, i.e., present the denotation of a class. In particular, dependence functions provide information for searching candidate tuples for correlation. A class may have parent classes for which it inherits attributes. For example, class `sportsCar`'s parent class is `Car`, so the class `sportsCar` inherits all the attributes in class `Car`.

Other than inheritance relationships, different classes may have value dependence on their properties. In our framework, dependence functions are used to indicate the value dependence among the different classes' properties. For example, we have three classes `ShippingDuration`, `Arrival` and `Departure`. In `ShippingDuration`, the attribute `duration` has a dependence function `minus(Arrival.timeStamp, Departure.timeStamp)`, where the predicate is `ShippingDuration.shippingID = Arrival.shippingID = Departure.shippingID`.

Based on dependence functions, a dependence tree can be constructed for each class. Assuming that the class C has a set of dependence functions F, a *dependence tree* can be generated as in Figure 1. There are three kinds of nodes in a dependence tree, namely *class node*, *operator node* and *dependant class node*. It should be noted that the depended class node may also have its own dependence tree (e.g., C_{11}). A class C's *complete dependence set* (denoted as $\mathbb{D}_C$) is defined as a collection of depended classes that can be used to calculate the value of the property. For example, the set $\{C_{11}, C_{12}, ..., C_{1m}\}$ is a complete dependence set of the class C's property p_1.

Once a class is defined, instances of the class can be created as objects (See 2). In the definition, the *ID* is the universal identifier for an object, while *V* gives values of attributes in the object.

Definition 2 (*Object*)**.** An object o is a 3-tuple$\langle ID, N_c, V\rangle$, o is an instance of a class C, where

- ID is the id of the object;
- N_c is the class name of C;
- $V = \{v_1, v_2, ..., v_n\}$, are values according to the attributes of the class C. For $v_i \in V$, v_i is a 2-tuple in form of $\langle N_p, V_p\rangle$, where N_p is the property name, V_p is the property value. □

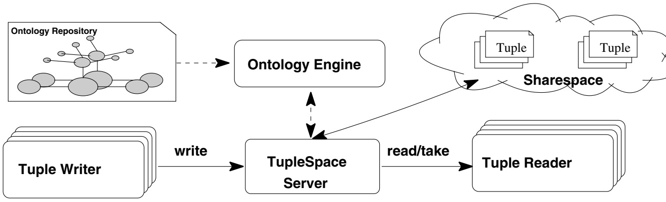

Fig. 2. Semantic Tuplespace System Architecture

Table 1. Notations

Notation	Definition
C	a class
$\mathbb{C}$	a set of classes
p_i	a class property
f_i	a dependency function
$\mathbb{D}_C$	a complete dependence set for class C
o	an object
$t(o_1, o_2, ..., o_n)$	a tuple
$\mathbb{C}_t$	the set consists of all t's field classes
$\mathbb{T}$	a set of tuples
$\mathbb{C}_\mathbb{T}$	the set consists of all field classes of tuples in $\mathbb{T}$
$\varphi(t_1, t_2, ..., t_n)$	a read/take template
$\mathbb{C}_\varphi$	the set consists of all the field classes required by template φ
q_i	a query predicate
t_i=$\langle C_i, q_i\rangle$	a formal field in template

2.2 System Architecture

Our semantic tuplespace system (see Figure 2) consists of an *ontology repository*, an *ontology engine*, *tuple writers*, *tuple readers*, *sharespace* and a *tuplespace server*. A tuple in the semantic tuplespace system is denoted as $t(o_1, o_2, ..., o_n)$, where each field in a tuple is an object o_i[2] and the class is C_i. An example of a tuple can be t_s(`sportsCarA`, `carInsuranceB`, `carFinanceC`), which contains three objects.

As in the traditional tuplespace system, the basic operations in semantic tuplespace include *write*, *read* and *take*. For tuple providers, the write operation is used to save tuples into the sharespace. For tuple consumers, the operations can be either read or take. The difference between read and take is that after a take the tuple is removed from the sharespace, while read leaves the tuple object in sharespace.

When performing a read/take operation, a template $\varphi(t_1, t_2, ..., t_n)$ that defines tuple matching conditions is specified . For each t_i in φ, it can be either *formal* or *non-formal* field. A formal field is specified as a pair $\langle C_i, q_i\rangle$, where the C_i specifies the class of the field and the q_i is a query predicate (a boolean expression of attributes in class C_i). A non-formal field is specified as $\langle o_i\rangle$ that indicates expecting an identical

[2] In the rest of the paper, we use term *object* and *field* interchangeably.

Table 2. Examples

Entity	Example
template	φ_s(⟨ `Car`, `Car.price.amount< 5000` ⟩, ⟨`carInsuranceB`⟩, ⟨`CarFinance`, `null`⟩)
candidate tuple	t(`sportsCarA`, `carInsuranceB`, `carFinanceC`)
tuple set	$\mathbb{T}_k$=$\{t_1, t_2\}$, where t_1(`sportsCarA`, `sportsCarInsuranceB`), t_2(`sportsCarA`, `carFinanceC`)
generated template for t_1	φ_1(⟨`SportsCar`, `SportsCar.price.amount<5000` ⟩, ⟨`carInsuranceB`⟩)
generated template for t_2	φ_2(⟨`SportsCar`, `SportsCar.price.amount<5000` ⟩, ⟨`CarFinance, null`⟩)
tuple set	$\mathbb{T}_f$=$\{t_1, t_2, t_3, t_4\}$, where t_1(`sportsCarA`, `licenceB`), t_2(`licenceB`, `carOwnerC`), t_3(`carOwnerC`, `carInsuranceD`), t_4(`sportsCarA`, `carFinanceE`)

object as o_i is contained in matched tuples. There are two options in read/take operation: *all* or *any*. Option `all` returns all the matched tuples, while option `any` only returns one of the matched tuples. In the rest of the paper, for sake of presentation, we only discuss option `all`; however, in our design, we support both options.

An example of template can be φ_s (see Table 2). In this example, the first field required by the template is an object of class `Car`, where the associated query predicate is `Car.price.amount<5000`. The second field is non-formal: object `carInsuranceB`, indicating that the tuples need to provide identical information as specified in the object. Actually, the non-formal field $\langle o_i \rangle$ can be converted to a formal field as $\langle C', \bigwedge_{j=0}^{n}(C'.p_j = o_i.p_j)\rangle$, where object o_i's class is C' that has n properties p_j. As such, in the rest of this paper, we only discuss the case of formal field.

2.3 Tuple Matching in Semantic Tuplespace System

By introducing ontologies into tuplespace system, other than *exact matching*, we extend the tuple matching algorithm with two extra steps: *semantic matching* and *correlation matching*. Therefore, three steps are involved in our matching algorithm:-

- **Step 1. Exact Matching.** The first step is to find exact matches, which returns tuples that have exactly the same field classes as the template;
- **Step 2. Semantic Matching.** The system searches tuples that have field classes which are semantically compatible with the template and delivers tuples if the tuples' contents can satisfy the filtering conditions;
- **Step 3. Correlation Matching.** The system searches a set of tuples and correlates them to one tuple, in order to match all required fields of the template.

It is worth noting that the type-based tuplespace system only performs step 1. The object-based tuplespace systems perform another step of matching that is based on object compatibility, which is different from semantic matching in step 2. In object-based tuplespace system, the object compatibility is deduced from the implementation of class hierarchy. In our semantic tuplespace systems, the relationships among the objects are

declaratively defined by ontologies. As such, the steps 2 and 3 are unique to our semantic tuplespace system. In this paper, we assume that both readers and writers use the same ontology for a domain. If a tuple writer and a tuple reader use different ontologies for a domain, then a common ontology can be created for both writer and reader. Detailed discussion on creating a common ontology is outside the scope of the paper. Therefore, by engineering ontologies, our system allows different services to exchange information using their native information format to construct tuples. The cost of engineering ontologies is much less than that of developing object adaptors for object-based tuplespace systmes as ontologies are declaratively defined. Further, ontologies are reusable. Details of semantic and correlation matching are presented in the following sections.

3 Semantic Matching

As an extension of object-based tuplespace system, semantic matching is used to determine whether a tuple in the sharespace satisfies a tuple retrieval request (read/take). The difference between object-based matching and semantic matching comes from the adopted approaches that determine the relation among the objects. As discussed earlier, object-based matching tuple matching is based on object compatibility, where the subclass relation is deduced from the implementation of class hierarchy. This requires all the tuplespace users to adopt the same implementation of class hierarchy. In our semantic matching, we adopt the notion of *semantic compatibility* (see 3), wherein the semantic knowledge of synonyms and subclasses can be declaratively defined in ontologies.

Definition 3 (*Semantic Compatibility*)**.** Class C_i is semantically compatible with class C_j, denoted as $C_i \stackrel{s}{=} C_j$, if in the ontology, either (i) C_i is the same as C_j (same name or synonym in an ontology) , or (ii) C_i is a superclass of C_j. □

By adopting the definition of semantic compatibility, we say a class C semantically belongs to a class set $\mathbb{C}$ (denoted as $C \in_s \mathbb{C}$) if $\exists C_i \in \mathbb{C}$, $C \stackrel{s}{=} C_i$. Using the notion of semantic compatibility, we define a *candidate tuple* (see 4) as a tuple that contains all the fields that are semantically compatible with the fields required by a read/take operation. In the definition, each of the fields of the tuple needs to be semantically compatible with the corresponding field of the template. For example (see Table 2), with regard to the template φ_s, the tuple t can provide all the fields required in φ_s since the first field `sportsCarA` "is a" `Car` (semantic compatibility) and the rest two fields are exactly matched. Therefore, t is a candidate tuple for φ_s.

Definition 4 (*Candidate Tuple*)**.** t is a tuple in tuplespace where $\mathbb{C}_t$ is the set that contains all the field classes in t; φ is the template for read or take operation, where the feild class set is $\mathbb{C}_\varphi$. t is a ***candidate tuple*** for φ iff: $\forall C_i \in \mathbb{C}_\varphi$, $C_i \in_s \mathbb{C}_t$. □

It should be noted that a candidate tuple may not be able to satisfy the filtering condition given in templates. Further examination of the contents of the tuple is required, in order to determinate whether the tuple should be delivered to tuple readers.

In our system, when inspecting the contents of tuples, in most cases, the tuplespace server needs to rewrite fields in the template, except when all the field classes in the candidate tuple are exactly the same as those of the template, i.e., $\mathbb{C}_t = \mathbb{C}_\varphi$. Therefore, each $\langle C_i, q_i \rangle$ in φ, assuming the class type of candidate tuple is C' for the corresponding field, should be rewritten as $\langle C', q_i^{'} \rangle$, where $q_i^{'}$ is transformed from q_i by replacing property references of class type C with C'.

4 Correlation Matching

As a further extension of object-based tuple matching, our system also enables correlating multiple tuples for a template. In the following subsections, we first present how to search a collection of tuples that are correlatable and are able to provide all compatible fields for read/take operation. This is followed by details on composing results from a collection of tuples.

4.1 Searching Correlatable Tuple Set for Read/Take Operation

In our framework, multiple tuples in the sharespace can be correlated to one that can provide all the necessary fields required by a template, wherein the correlation can be done by the join operator. Correlation can be either based on common fields and/or attribute dependence functions. In this subsection, we discuss the case of field-based correlation first, and then illustrate the case of attribute dependence function correlation.

Field-Based Correlation. Obviously, multiple tuples can be correlated using the join operator to one if they contain same field. For example, two tuples t_1 and t_2 in $\mathbb{T}_k$ (see Table 2) can be correlated using the join operator as they both have field `sportsCarA`. Therefore, when the tuplespace server performs the correlation matching, in order to compose tuples that can provide all the fields that are required by the template, it first searches a *key-based correlation tuple set*, i.e., a set of tuples that are correlatable by a key field that is specified by the template and can provide all the fields required by the template. The formal definition of key-based correlation tuple set is as follows.

Definition 5 (*Key-based Correlation Tuple Set S_{kc}*). $\mathbb{T}$ ($\mathbb{T} = \{t_1, t_2, ..., t_n\}$) is a set of tuples in tuplespace, $\mathbb{C}_{t_i}$ is the set that consists of all the field classes in tuple t_i and $\mathbb{C}_\mathbb{T}$($\mathbb{C}_\mathbb{T} = \cup_{i=1}^{n} \mathbb{C}_{t_i}$) is aggregation of all the field classes in $\mathbb{T}$; φ is the template for read/take operation, C_k is the key field's class type and $\mathbb{C}_\varphi$ is the set that consists of all the field classes of φ. $\mathbb{T}$ is a ***Key-based Correlation Tuple Set*** of φ iff:

1. $\forall C \in \mathbb{C}_\varphi, C \in_s \mathbb{C}_\mathbb{T}$;
2. $\forall \mathbb{C}_{t_i}, \exists C_k^{'} \in \mathbb{C}_{t_i}, C_k \stackrel{s}{=} C_k^{'}$, and $o_1^k = o_2^k = ... = o_n^k$, where o_i^k is the field with class $C_k^{'}$ in t_i;
3. $\forall \mathbb{C}_{t_i}, \exists C, C \in (\mathbb{C}_{t_i} - (\cup_{j=1}^{i-1} \mathbb{C}_{t_j} \bigcup \cup_{j=i+1}^{n} \mathbb{C}_{t_j}))$ and $C \in_s \mathbb{C}_\varphi$. $\square$

In this definition, three conditions need to be satisfied when considering a set of tuples as a correlation tuple set for a read/take template: (i) Condition (1) indicates for

each field class required by the template, there is at least one tuple that contains a compatible field class, which is a necessary condition of the definition. (ii) Condition (2) implies all the field classes are correlatable by the key field. (iii) Condition (3) evinces any tuples in the set contributes at least one unique field. It should be noted that condition (2) and (3) are the sufficient conditions for the definition. Using above example, the aggregation of t_1 and t_2 provides all the required fields in template, which satisfy condition (1), and they can be correlated as they share the field `sportsCarA` that is the descendant for the key field `Car` in template φ_s. Also, t_1 (resp. t_2) provides unique field `carInsuranceB` (resp. `carFinanceC`). Therefore, t_1 and t_2 compose a key-based correlation tuple set for the template.

Actually, by releasing the constraint that correlating is based on key field only, our system enables more generic tuple correlation, wherein tuple correlations can be based on any fields. In such a generic correlation, we adopt the notion of `Correlatable Class` (see 6). In this definition, two field classes are correlatable in a set of tuples if either they appear in the same tuple, or when these two classes do not appear in the same tuple and belong to two tuples t_x and t_y respectively, then either (i) t_x and t_y at least have one field that is identical; or (ii) there are a sequence tuples in the set that are correlatable "step by step" and aiming for correlating t_x and t_y in the end. Actually, if we consider t_x and t_y are `entities` in ER model, then these tuples between t_x and t_y in the sequence are `relationships`: in order to joint two entities without common attributes, a collection of relationships $[t_{x+1}, t_{x+2}, ... t_{y-1}]$ are required. For example, class `SportsCar` and `CarInsurance` are correlatable in $\mathbb{T}_f$ (see Table 2), as class `SportsCar` and `CarInsurance` appear in t_1 and t_3 respectively; and t_2 is considered as a relationship to bridge `SportsCar` and `CarInsurance`.

Definition 6 (*Correlatable Class*). *Class C_i, C_j are correlatable in tuple set $\mathbb{T}$ ($\mathbb{T} = \{t_1, t_2, ..., t_n\}$), iff either*

- *C_i and C_j appear in same tuple (i.e., $\exists t_x \in \mathbb{T}$, both C_i and $C_j \in \mathbb{C}_{t_x}$); or*
- *C_i and C_j do not appear in same tuple (i.e., $\nexists t \in \mathbb{T}$, where C_i and $C_j \in \mathbb{C}_t$), then $\exists t_x, t_y \in \mathbb{T}$, $x \neq y$, $C_i \in \mathbb{C}_{t_x}$, $C_j \in \mathbb{C}_{t_y}$, and either:*
  - *$\exists o_x$ from t_x and $\exists o_y$ from t_y, $o_x = o_y$; or*
  - *there is a correlation tuples sequence $[t_x, t_{x+1}, t_{x+2}, ... t_{y-1}, t_y]$ in $\mathbb{T}$, and for any t_i, t_{i+1} in the sequence, $\exists$ o_i from t_i and $\exists o_{i+1}$ from t_{i+1}, so that $o_i = o_{i+1}$.* □

Definition 7 (*Field-based Correlation Tuple Set S_{fc}*). $\mathbb{T}$ ($\mathbb{T} = \{t_1, t_2, ..., t_n\}$) is a set of tuples in tuplespace, $\mathbb{C}_{t_i}$ is the set that consists of all the field classes in tuple t_i and $\mathbb{C}_{\mathbb{T}}$($\mathbb{C}_{\mathbb{T}} = \cup_{i=1}^{n} \mathbb{C}_{t_i}$) is aggregation of all the field classes in $\mathbb{T}$; φ is the template for read/take operation, and $\mathbb{C}_\varphi$ is the set that consists of all the field classes of φ. $\mathbb{T}$ is a ***Field-based Correlation Tuple Set*** of φ iff:

1. $\forall C \in \mathbb{C}_\varphi, C \in_s \mathbb{C}_{\mathbb{T}}$;
2. for $\forall$ $C_i^{'}, C_j^{'} \in \mathbb{C}_\varphi$, $i \neq j$, $\exists$ $C_i, C_j \in \mathbb{C}_{\mathbb{T}}$, $C_i^{'} \stackrel{s}{=} C_i$, $C_j^{'} \stackrel{s}{=} C_j$, and C_i and C_j are correlatable in $\mathbb{T}$;
3. $\forall t_i \in \mathbb{T}$, at lease one of the following is true:
   - $\exists C \in (\mathbb{C}_{t_i} - (\cup_{j=1}^{i-1} \mathbb{C}_{t_j} \bigcup \cup_{j=i+1}^{n} \mathbb{C}_{t_j}))$, $C \in_s \mathbb{C}_\varphi$;
   - t_i appears in tuple consequences in condition (2) of this definition. □

Using the notion of correlatable class, we can define the concept of *Field-based Correlation Tuple Set* (see 7). In the definition, there are also three conditions that need to be satisfied when considering a set of tuples as a correlation tuple set for a read/take template: (i)The same as key-based correlation, condition (1) indicates for each field class required by the template. (ii) Different from key-based correlation, instead, Condition (2) implies correlation can be on any fields. (iii) Condition (3) evinces any tuples in the set contributes at least one unique field, either contributes to the required fields by the template, or appearers in tuple sequence for correlation.

Attribute-Dependence Correlation. Other than field-based, multiple tuples can be correlated using dependence functions, in case some required fields can not be provided by any available tuples. Assuming that an absent field's class C_i has a dependence function, the tuplespace server can compute the value for the absent field from the tuples that provide elements in the dependence set. For example, if the class type `ShippingDuration` is required by the template but not provided by any tuples,

as `ShippingDuration`'s dependence set is {`Departure, Arrival`}, the system can search tuples that contain `Departure` or/and `Arrival` and correlate these tuples and compute the value for `ShippingDuration`. Again, we first limited the correlation on key field only, wherein *Key-based Attribute-dependence Correlation Tuple Set* can be defined as:

Definition 8 **(***Key-based Attribute-dependence Correlation Tuple Set* S_{ka}**).** $\mathbb{T}$ ($\mathbb{T} = \{t_1, t_2, ..., t_n\}$) *is a set of tuples in tuplespace,* $\mathbb{C}_{t_i}$ *is the set that consists of all the field classes in tuple* t_i *and* $\mathbb{C}_{\mathbb{T}}$ ($\mathbb{C}_{\mathbb{T}} = \cup_{i=1}^{n} \mathbb{C}_{t_i}$) *is aggregation of all the field classes in* $\mathbb{T}$*;* φ *is the template for read/take operation, the key field's class is* C_k *and* $\mathbb{C}_\varphi$ *is the set that consists of all the field classes in* φ*.* $\mathbb{T}$ *is an* ***Key-based Attribute-dependence Correlation Tuple Set*** *of the template* φ *iff:*

1. $\forall C_i \in \mathbb{C}_\varphi$, *either*
   - *if* $C_i \in_s \mathbb{C}_{\mathbb{T}}$, *i.e.,* $\exists\, C_i' \in \mathbb{C}_{\mathbb{T}}$, $C_i \overset{s}{=} C_i'$*; or*
   - *if* $C_i \notin_s \mathbb{C}_{\mathbb{T}}$, *then* $\mathbb{C}_{\mathbb{T}}$ *contains a complete dependence set* $\mathbb{D}_{C_i}$ *of* C_i*.*
2. $\forall \mathbb{C}_{t_i}$, $\exists C_k' \in \mathbb{C}_{t_i}$, $C_k \overset{s}{=} C_k'$, *and* $o_1^k = o_2^k = ... = o_n^k$, *where* o_i^k *is the field with class* C_k' *in* t_i*;*
3. $\forall t_i \in \mathbb{T}$, *at lease one of the following is true:*
   - $\exists C \in (\mathbb{C}_{t_i} - (\cup_{j=1}^{i-1} \mathbb{C}_{t_j} \bigcup \cup_{j=i+1}^{n} \mathbb{C}_{t_j}))$, $C \in_s \mathbb{C}_\varphi$ *or* $C \in \mathbb{D}_{C_i}$*;*
   - t_i *appears in tuple consequences in condition (2) of this definition.* □

In condition (1) of above definition, unlike field-based correlation tuple set, a field required by the template may not appear in any tuple, however, its properties can be computed using dependence functions (See 2). Like field-based correlation in tuple set, the condition (2) concerns whether tuples can be correlated by the key field. The condition (3) states that each tuple in the set contributes at least one unique attribute. Again, we can release the constraint that correlation is based on key-field only . Therefore, the more generic *Attribute-dependence Correlation Tuple Set* can be defined (see 9). In particular, the condition 2 of the definition indicates that correlation can be done based on any fields.

Definition 9 (*Attribute-dependence Correlation Tuple Set* S_{ac}**).** $\mathbb{T}$ ($\mathbb{T} = \{t_1, t_2, ..., t_n\}$) *is a set of tuples in tuplespace,* $\mathbb{C}_{t_i}$ *is the set that consists of all the field classes in tuple* t_i *and* $\mathbb{C}_{\mathbb{T}}$ *(*$\mathbb{C}_{\mathbb{T}} = \cup_{i=1}^{n} \mathbb{C}_{t_i}$*) is aggregation of all the field classes in* $\mathbb{T}$*;* φ *is the template for read/take operation;* $\mathbb{C}_{\varphi}$ *is the set that consists of all the field classes in* φ*.* $\mathbb{T}$ *is an* ***Attribute-dependence Correlation Tuple Set*** *of the template* φ *iff:*

1. $\forall C_i \in \mathbb{C}_{\varphi}$*, either*
   - *if* $C_i \in_s \mathbb{C}_{\mathbb{T}}$*, i.e.,* $\exists\, C_i^{'} \in \mathbb{C}_{\mathbb{T}}$*,* $C_i \stackrel{s}{=} C_i^{'}$*; or*
   - *if* $C_i \notin_s \mathbb{C}_{\mathbb{T}}$*, then* $\mathbb{C}_{\mathbb{T}}$ *contains a complete dependence set* $\mathbb{D}_{C_i}$ *of* C_i*.*
2. *Assuming* $\mathbb{C}^{'}$ *is the class set for all the* $C_i^{'}$ *in condition 1 of this definition, also assuming* $\mathbb{D} = \bigcup \mathbb{D}_{C_i}$ *for all* $C_i \notin_s \mathbb{C}_{\mathbb{T}}$*, and* $\mathbb{C} = \mathbb{C}^{'} \bigcup \mathbb{D}$*, then for* $\forall\, C_i, C_j \in \mathbb{C}$*,* C_i *and* C_j *are correlatable in* $\mathbb{T}$*;*
3. $\forall t_i \in \mathbb{T}$*, at lease one of the following is true:*
   - $\exists C \in (\mathbb{C}_{t_i} - (\cup_{j=1}^{i-1} \mathbb{C}_{t_j} \bigcup \cup_{j=i+1}^{n} \mathbb{C}_{t_j}))$*,* $C \in_s \mathbb{C}_{\varphi}$ *or* $C \in \mathbb{D}_{C_i}$*;*
   - t_i *appears in tuple consequences in condition (2) of this definition.* □

4.2 Relationship Among Four Kinds of Correlation Tuple Sets

The relationship among the above four kinds of correlation tuples sets is shown in Figure 3. In particular, the relationship can be summarized as:

- $\{S_{kc}\} \subseteq \{S_{fc}\}$
  Proof: Condition (1) in S_{kc} and S_{fc} are same. Further, the tuple set can satisfy condition (2) and (3) of S_{kc} can also satisfy condition (2) and (3) in S_{fc}.
- $\{S_{fc}\} \subseteq \{S_{ac}\}$
  Proof: Condition (1) in S_{fc} is the first situation of condition (1) in S_{ac}. Condition (2) in in S_{fc} is same as Condition (2) in S_{ac} when $\mathbb{D} = \varnothing$, i.e., all the field classes in the template do not have any attribute dependence function. Condition (3) in both S_{fc} and S_{ac} is same.
- $\{S_{kc}\} \subseteq \{S_{ka}\}$
  Proof: Condition (1) in S_{kc} is the first situation of condition (1) in S_{ka}. Condition (2) in in S_{kc} is same as Condition (2) in S_{ka} when $\mathbb{D} = \varnothing$, i.e., all the field classes in the template do not have any attribute dependence function. Condition (3) in S_{kc} is the first situation in condition (3) in S_{ka}.
- $\{S_{ka}\} \subseteq \{S_{ac}\}$
  Proof: Condition (1) and (3) in S_{ka} and S_{ac} are same. Further, the tuple set can satisfy condition (2) of S_{ka} can also satisfy condition (2) in $\{S_{ac}\}$ as $o_1^k = o_2^k = ... = o_n^k$ can guarantee all the required field classes are correlatable in tuple set.

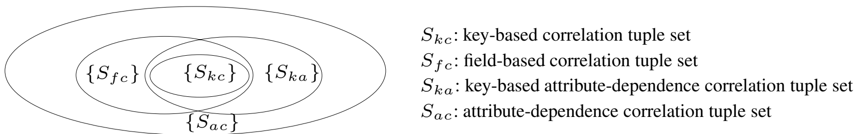

Fig. 3. Relationship Among Four Kinds of Correlation Tuple Sets

4.3 Template Generation for Correlation Matching

From the above discussion we know that both types of correlatable tuple sets can only guarantee that the fields required for the template can be provided or computed. However, further inspection of the contents of tuples is required, in order to determine whether the filtering conditions given in templates can be satisfied. In our solution, this is realized by generating a template for each tuple in the set and then using the generated templates to inspect the contents of each tuple individually.

Assuming there are n tuples t_i in the correlation set $\mathbb{T}$ ($t_i \in \mathbb{T}$), We distinguish two types of fields in $\mathbb{T}$: *unique* and *non-unique* fields: unique fields are the fields that are required by the template φ and only appear in one tuple in the tuple set ($\mathbb{C}^u_{\mathbb{T}}$ denotes the collection of all the unique fields), while non-unique fields appear in more than one tuple in the set. From the definition of correlation tuple set, $\forall C \in \mathbb{C}^u_{\mathbb{T}}$, $\exists C' \in \mathbb{C}_{\varphi}$, C' either is the same as C or super class of C. Therefore, for each $\langle C', q' \rangle$ in a template, in the case of $C' = C$, then in the template φ_i for tuple t_i, $\langle C', q' \rangle$ is used without any changes; while in the case of C' is super class of C, $\langle C', q' \rangle$ need to be transformed to $\langle C, q \rangle$, where query predicate q is transformed from q' by replacing referenced property of C' with property in C.

For example, considering the tuple set $\mathbb{T}_k$ for the template φ_s, two temples φ_1 and φ_2 are generated respectively (see Table 2). In particular, the query predicate SportsCar`.price.amount<5000` in φ_1 is transformed from Car`.price.amount<5000` in φ, where Car is replaced by SportsCar.

Once a template φ_i is generated for each t_i in $\mathbb{T}$, the tuplespace server needs to test the query predicates for fields in each template and correlate tuples. In the case of field-based correlation tuple set, when inspecting the tuple using the generated template, the false result of query predicate on any tuple in the set will result in discarding the whole tuple set from further correlation processing. After testing all templates, if the tuple set is not discarded, the tuple set is correlated to one tuple. Again, we differentiate two different kinds of fields. For unique field, it can be selected from a tuple. For non-unique field, the tuplespace server prefers a tuple which has same type of field as template required. By selecting each field required by the template, a tuple is created and delivered to the reader.

In the case of attribute-dependence correlation tuple set, another step is required on the correlated tuple: applying the dependence functions to compute the field value and testing the associated query predicate to determinate whether the generated tuple should be delivered to the reader.

5 Implementation Aspects

In this section, we discuss the implementation of the proposed tuplespace server (see Figure 4), which consists of four components: *Write Manager*, *Runtime Store*, *Persistent Datastore* and *Read/Take Manager*.

Our tuplespace server supports tuple correlation. This requires the tuplespace server to persist tuples when they are writing into sharespace, for possible correlation operation on them thereafter, as it is unlikely that the main memory can store all the tuples in the sharespace. Further, persistent support also allows tuplespace server restores from

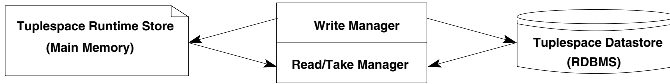

Fig. 4. System Architecture of Tuplespace Server

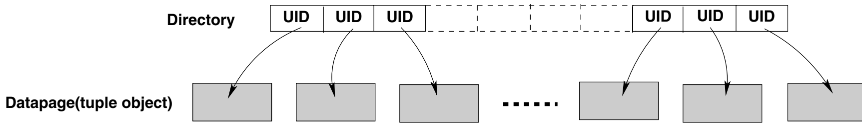

Fig. 5. Hash Index in Runtime Store

runtime failure, which is a key requirement for mission critical applications. Therefore, in our design, the Tuple Writer manages both Runtime Store in main memory and Persistent Datastore in Relational Database. When Tuple Writer receives a write tuple request from users, it saves the tuple object in both the Runtime Store and the Persistent Datastore. In case the main memory is full, it needs to remove some tuples from Runtime Store, wherein `First In First Out` update algorithm is adopted. In our design, tuples in the Runtime Store as objects have unique object IDs. As the runtime store is considered as a cache for the tuplespace datastore, we create a *tuple ID-based* hash index, where the unique object ID is used to locate the tuple object. Therefore, when the tuple writer receives a tuple, it saves the tuple with the unique object ID, and then invokes hash functions to update the hash index. When the tuple writer saves a tuple object in runtime store, it also persists the tuple object in tuplespace Datastore. This cache improves the system performance on retrieving tuple contents when tuple UIDs are identified.

The datastore provides persistent storage of tuples. When considering the implementation of datastore, the intuitive choice is adopting object store (i.e., persist tuples as objects). However, it is very costly when inspecting tuples' contents for tuple matching: entire tuple objects need to be deserialized in the memory. In fact, in most cases, tuple matching may only concern some attributes of tuples. For the sake of performance and scalability, instead of adopting object store, relation database is used to implement Persistent Datastore. Therefore, when conducting tuple matching, the inspection can only focus on the attributes that are concerted by the templates, without deserialization of entire tuple objects.

When adopting relational approach to persist tuples, mapping between tuple objects and relation tables is required. As user operations on tuples do not explicitly declare the data schema of the tuple (i.e., declaration of tuple schema is not required by the tuplespace system), a tuple can not be stored as a record in a predefined table. In our solution, the tuplespace server separates the data organization of tuple and contents of tuples (see Figure 6), wherein one table `FieldTypes` is used to store the class type information for each field in tuples, while another table `TupleValues` is used to store the contents of tuples. It should be noted that both class type information and the content of the tuples are stored vertically in these tables. In particular, for table

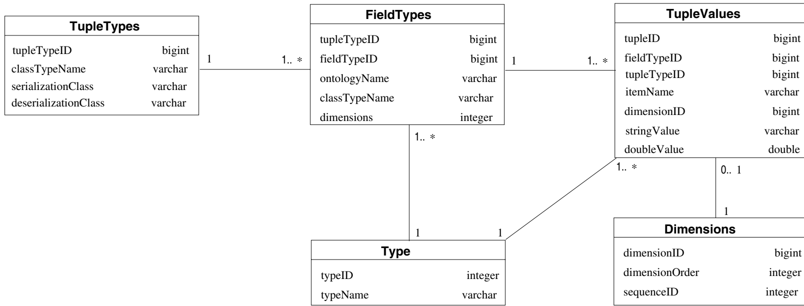

Fig. 6. ER diagram for Persistent Datastore

FieldTypes, each field in a tuple occupies a row. For each tuple in tuplespace a unique tupleTypeID is assigned for each type of tuple. In table TupleValues, each elementary element in a field has a record in the table and tupleID is unique for each tuple in tuplespace. Using tupleID and fieldTypeID, the records in the table can be correlated to individual tuples. Table Dimensions (D for short) is used to store the dimension information when there exists any array type of data elements in fields. By specifying dimensionOrder and sequenceID, the datastore can store any dimension array of data in a tuple. Further, the table Types gives type information in tuplespace.

The Read/Take Manager handles tuple read/take requests from users. When it receives read/take requests, it searches for a single tuple that can match the template first. In case there are no single tuple matching the template or users required, the Read/Take Manager searches a correlation tuple set for the temple. In our solution, both semantic and correlation matching is done by generating queries on persistent data store. Details on design of query generation are omitted due to space reasons.

6 Related Work

An effort to provide semantic support in communication paradigm is given in [15], wherein we introduce ontologies into the publish/subscribe systems to understand event contents. This relaxes the constraint in prior content-based pub/sub systems that publishers and subscribers must share the same event schemas. It supports semantic-based, automatic event correlations of multiple event sources for subscriptions, which also overcomes the limitation of relational publish/subscribe systems [8] that requires event consumers to explicitly specify the correlation of event sources. In this paper, we apply the same idea to the tuplespace system, i.e., providing flexibility and adaptability in communication paradigm by leveraging data semantics. Unlike publish/subscribe systems, the tuplespace system does not require the explicit declaration of the data schema when reading and writing tuples, which imposes new challenges when introducing semantic support.

The tuplespace system is a very active area of research and development. Early tuplespace systems [4,7] can be considered as a software implementation of a shared

distributed memory. The sharespace appears as a single shared blackboard where tuples can be deposited. Readers are notified when the values of tuples match templates. In this way information can be shared and tuples can be passed among services. As a further development for supporting coordination among services, reactivity was added into tuplespace [3], where local policies could be specified for interactions among the tuple readers and writers. However, in the above systems, expression power of specifying the matching template is very limited for both data type and exactly value matching.

ObjectSpace [12] and T Space [10] added object-orientation to the tuplespace system, wherein a template and a tuple match if the type of the tuple is an instance of the type in the template. The limitation is that object-compatibility assumes both tuple writers and readers adhere to the same implementation of class hierarchy. In our semantic tuplespace system, we externalize the semantic of tuples, wherein ontology is used to understand the content of tuples in matching algorithm. On the other aspect, in order to enhance the tuple retrieval power, PLinda [1] added database functionalities (query predicates, join operator and transaction) into tuplespace systems. In our semantic tuplespace, not only are the database functionalities fully supported, but the join operation is also transparent to tuple readers when correlating multiple tuples for tuple matching.

Triple Space [6,2] provides an asynchronous communication mechanism that supports the four types of autonomy: time, space, reference, and data schema. The data schema is implemented using RDF [13] to understand the communication contents, which is similar to semantic matching in semantic tuplespace system. It should be noted that with the persistent data store, our system also supports these four types autonomy. In additional, our system provides correlation autonomy, i.e., automatic tuple correlation.

An initial effort to provide semantic support for tuplespace matching is given in sTuple [9], which relaxes the constraints in object-based tuplespace that readers and writers must share the same implementation of class hierarchy. However, it only considers one tuple and one template matching. This is similar to the case of semantic matching in our system. Our system proposes a comprehensive schematic tuplespace system. In particular, it supports semantic-based, automatic correlations of multiple tuples for tuple matching, which also overcomes the limitation of PLinda system that requires tuple readers to explicitly specify the correlation of tuples.

Semantic matching is widely adopted to solve the service matching problem in the semantic Web [5,14]. The basic idea of semantic matching is to determine the semantic distance between co-existent terms within shared ontologies, where service queries and descriptions are based on pre-defined schema. However, the semantic matching that enables tuple routing in our tuplespace system is conducted without the user's declaration of data schema when writing and reading tuples.

7 Conclusion

In this paper, we propose a semantic tuplespace system, which is another step forward in the development of current tuplespace systems. We introduce semantics to understand the tuple contents. Our system not only considers single tuple for read/take operation, but also automatically correlates multiple tuples using relational operators based on

templates. Unlike object-based tuplespace systems, the tuple correlation in our system is transparent to the tuple reader. We argue that the proposed tuplespace system is essential to enable cooperative service communication in service-oriented computing. Our future work includes optimization of semantic tuple matching and tuple correlation, and a scalability and reliability study of the system.

References

1. B. Anderson and D. Shasha. Persistent Linda: Linda + transactions + query processing, 1991.
2. C. Bussler. A minimal triple space computing architecture. In *2nd WSMO Implementation Workshop, Innsbruck, Austria, June 2005*.
3. G. Cabri, L. Leonardi, and F. Zambonelli. Reactive tuple spaces for mobile agent coordination. *Lecture Notes in Computer Science*, 1477, 1998.
4. N. Carriero and D. Gelernter. Linda in context. *Commun. ACM*, 32(4):444–458, 1989.
5. D. Chakraborty, F. Perich, S. Avancha, and A. Joshi. Dreggie: Semantic service discovery for m-commerce applications, 2001.
6. D. Fensel. Triple-space computing: Semantic web services based on persistent publication of information. In *IFIP Int'l Conf. on Intelligence in Communication Systems 2004*, pages 43–53.
7. Y. S. Gutfreund, J. Nicol, R. Sasnett, and V. Phuah. Wwwinda: An orchestration service for www browsers and accessories. In *WWW Conference '94: Mosaic and the Web*, 1994.
8. Y. Jin and R. Strom. Relational subscription middleware for internet-scale publish-subscribe. In *2nd international workshop on Distributed event-based systems, San Diego, California*, pages 1–8, 2003.
9. D. Khushraj, O. Lassila, and T. Finin. sTuples: Semantic Tuple Spaces. In *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, Boston, Massachussets, USA*, August 22 - 26, 2004.
10. T. J. Lehman, S. W. McLaughry, and P. Wycko. T spaces: The next wave. In *HICSS*, 1999.
11. OWL, 2005. http://www.w3.org/TR/owl-ref/.
12. A. Polze. Using the Object Space: a Distributed Parallel Make. In *The 4th IEEE Workshop on Future Trends of Distributed Computing Systems, Lisbon, September*, 1993.
13. RDF Primer, W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-primer/.
14. K. Sycara, S. Wido, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace, 2002.
15. L. Zeng and H. Lei. A semantic publish/subscribe system. In *CEC-EAST '04: Proceedings of the E-Commerce Technology for Dynamic E-Business, IEEE International Conference on (CEC-East'04)*, pages 32–39, Washington, DC, USA, 2004. IEEE Computer Society.