Electronic Communications of the EASST
Volume 19 (2009)

Proceedings of the
Second International DisCoTec Workshop on
Context-aware Adaptation Mechanisms for
Pervasive and Ubiquitous Services
(CAMPUS 2009)

TOTAM: Scoped Tuples for the Ambient

Christophe Scholliers, Elisa Gonzalez Boix and Wolfgang De Meuter

15 pages

Guest Editors: Romain Rouvoy, Michael Wagner
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

TOTAM: Scoped Tuples for the Ambient

Christophe Scholliers[1], **Elisa Gonzalez Boix**[2] **and Wolfgang De Meuter**[3]

[1] cfscholl@vub.ac.be, [2] egonzale@vub.ac.be, [3] wdmeuter@vub.ac.be

Programming Technology Lab
Vrije Universiteit Brussels, Belgium

Abstract: Coordination of mobile applications posses a number of issues. Devices should be able to communicate with each other without being connected with each other at the same time while maintaining privacy and limited network traffic. Current tuple based approaches solve these issues partially but none of them solves *all* of them. We propose a novel tuple space-based approach where tuple spaces are annotated with tuple space descriptors used to determine the scope of a tuple. The novelty of our approach lies in the use of these tuple space descriptors to determine that a tuple should be propagated *before* it is transmitted. This enhances privacy and decreases the burden on the network traffic in a wide range of applications.

Keywords: mobile ad hoc networks, distributed computing, tuple spaces, scopes

1 Introduction

Advances in wireless communication technology and the increasing minutarization of computing devices have given rise to a growing body of research in pervasive computing which deals with *mobile ad hoc networks*. Such networks are composed of *mobile* devices which are connected by *wireless* communication links with a limited communication range [MCE02].

Developing applications for mobile networks is substantially complicated because of two discriminating properties which clearly set mobile networks apart from traditional, fixed computer networks [VMG+07]: nodes in the network only have intermittent connectivity (due to the limited communication range of wireless technology combined with the mobility of the devices) and applications need to discover and collaborate with one another in an ad hoc manner without relying on any centralized coordination facility. As a result, applications must deal with a highly dynamic environment and adapt their behaviour to changes on their physical and computation context. Mobile ad hoc applications range from collaborative text editors or file sharing services, to slightly more futuristic applications where e.g. cities are equipped with wireless base stations allowing commuters to obtain traffic information of their itinerary.

Tuple-space based middleware has shown to provide an appropriate computational model for dealing with such characteristics [MCE02]. Several tuple space-based coordination middleware have been specially developed for mobile computing (including LIME [MPR01], L2imbo [DFWB98] and TOTA[MZ04]). However, none of those approaches provides a *dynamic* scoping mechanism which limits the *physical* transportation of tuples. TOTA is one of the most dynamic tuple-based solutions for mobile networks: rather than merging local tuple spaces upon network connection, tuples themselves decide how to propagate from a tuple space to another. As TOTA tuples allow to express application-specific propagation rules, they can be exploited to achieve context-awareness in an adaptive way, making this model ideal for a mobile ad hoc setting. In TOTA, the participation of intermediate tuple spaces is exploited to reach a target tuple space. One important observation is that as all devices can potentially access all information, information cannot be hidden or scoped. Some mobile applications may require privacy in their communication, e.g. the payment of a product should be a private activity between a vending machine and a cellular phone. Not only may requiring the participation of all tuple spaces be unacceptable for certain applications, it also generates network flooding and has performance repercussions on mobile devices which are likely to have scarce resources, such as limited battery power.

In this paper, we propose a novel framework called TOTAM (”Tuples on the Ambient”) which extends TOTA with *dynamically* scoped tuples. We provide the programmer with means to scope the tuples themselves, i.e the tuples can dynamically adjust their scope as they hop from location to location. This scope is determined before the tuple is transmitted, thus allowing the programmer to prevent the physical transportation of tuples to devices which are not targeted. Scoped tuples have a number of benefits: tuples carry the definition of the target tuple spaces enhancing privacy and avoiding unnecessary exchange of tuples.

2 Motivation

The main motivation for TOTAM stems from the characteristics that set mobile networks apart from traditional computer networks. We first introduce a simple yet representative scenario that exhibits a number of issues that have to be dealt with by the distributed computational model.

2.1 Scenario: Ambient Game

Consider a multi player game running on mobile devices where users can use their PDA’s to chase dangerous (virtual) gangsters around the campus (in the outdoors). Players are organized in teams which determine their role in the game. For example, the blue team are policemen and the red team are gangsters. Players have a representation of all nearby team members on their PDA which allows them to keep track of their location (e.g. using GPS coordinates) and to orient themselves in the campus in order to coordinate their movements. For example, proximate team players can send messages to each other in order to decide on a group strategy to defeat the other team. Additionally, players carry (virtual) objects which can be used to damage members of the opposite team. For example, gangsters could throw bombs to the nearby policemen.

In order to implement the scenario described we identify five important issues which are derived from the distinguishing hardware characteristics of mobile ad hoc networks:

Discovery Players need to detect and interact with other players not known beforehand. In a mobile network where devices spontaneously join and leave the network unannounced, relying on a centralized infrastructure for service discovery may not be possible when they meet out in the open and setup an ad hoc network. As a result, devices need to discover each other spontaneously in an ad hoc manner.

Context-awareness Players need to react to frequent changes in the environment, such as change of location, or the appearance and disappearance of other players at any time. Due to the very nature of mobile networks, applications are exposed to an extremely dynamic context requiring contextual adaptations of their behaviour.

Communication Players should not be excluded from the team coordination when they move out of communication range. Upon reconnection, players expect to resume their computation and receive information about team members positions or the strategy being followed. Such information can also be provided by nearby team members which the player did not meet yet. In other words, two players do not need to be connected with each other at any time to coordinate their movements and positions as information can be carried around by other team members.

Privacy The information shared amongst members of the same team (e.g their positions or the agreed strategy) should not be accessed by the other team players which could use it to their own advantage. Non-targeted or non-authorized devices should not be exposed to sensitive information to avoid endangering privacy.

Scarce resources PDA's have scarce resources, such as limited battery power and limited communication range. This may require applications to achieve their goal with limited computational and communication efforts, e.g. minimizing network traffic.

We consider the ambient game application presented in this section to be an illustrative example which exhibits a set of key issues that are typical in collaborative ad hoc networking applications.

2.2 Tuple space-based middleware

None of the existing tuple space-based approaches deals with all the issues that we have identified for the development of collaborative ad hoc networking applications. In this section we briefly explain the concept of a tuple space and explain why current tuple space-based approaches do not address all of the issues.

Tuple spaces were first introduced in the coordination language Linda [Gel85]. A tuple space is a shared associative memory used by processes to communicate. Processes can post and read tuples using three basic operations: `out` to insert a tuple into the tuple space, `in` to remove a tuple from the tuple space and `rd` to check if a tuple is present in the tuple space (without removing it). Tuples are anonymous and are extracted from the tuple space by means of pattern matching on the tuple contents. Tuple space communication is decoupled both in space and time: processes do not have to know each other beforehand nor to be available at the same time since tuples can be inserted and extracted independently. These forms of decoupling address the discovery and communication issues making the model suitable for mobile networks. However, maintaining a globally shared tuple space is not feasible in a mobile setting.

Some adaptations of tuple spaces targeting the mobile environment, such as LIME, have extended this model with the notion of *federated tuple spaces*. In this model every node in the network keeps a local tuple space. The tuples in the local tuple space of all devices in range are conceptually merged into a federated tuple space. Nodes can post and read tuples from this federated tuple space by means of the typical tuple space operations. When devices move out of range their tuples are no longer shared and removed from the federated tuple space. LIME supports physical context-awareness based on connectivity. A disadvantage of LIME is that tuples can only be exchanged when the communication partner that issued a tuple space operation is in range of the device offering the requested tuple.

TOTA improves on this model by allowing tuples to be replicated from node to node loosening the restriction that the sender and the receiver of a tuple have to be connected at the same time. Rather than merging local tuple spaces when devices discover, tuples are equipped with a *propagation rule* that determines how a tuple hops from one tuple space to another one. The propagation rule can also change the tuple information, thus tuples can be dynamically adapted while getting propagated in the network. These propagation rules are crucial to provide programmers with a flexible mechanism to express more elaborated kinds of information sharing than simple merging of information supported by LIME. Tuples can be exploited to achieve context-awareness based not only on connectivity but also on semantic information. Tuples in TOTA are sent to all communication partners in range. Upon arrival at the receiver side, the tuple itself decides whether it has to be stored in that tuple space. By transmitting tuples potential malicious or non-intended users are provided with sensitive information. Not only does sending all tuples blindly to all communication partners in range raise privacy issues, it also creates a network traffic overhead.

In order to deal with these issues, LIME-based systems have introduced some notion of scope on the tuple space [DFWB98, MW00, IA06, CMMP06, SJ07]. However, to the best of our knowledge, none of those approaches (1) provide the fine grained specification of propagations rules found in TOTA and (2) have support to dynamically change the scope of a tuple. In scoped-based approaches tuples are inserted in a certain scope which can not be changed after its publication. In order to change it, tuples have to be removed and inserted again in the new scope.

In summary, we observe that LIME-based approaches deal with the discovery, (physical) context-awareness, privacy and scarce-resource issues distilled from the scenario introduced in the previous section. However, they fall short of the communication issue as they do not allow tuples to be transported through multiple hops while the sender and receiver devices are not connected at the same time. On the other hand, TOTA deals with this communication issue as it provides flexible tuples which can hop from device to device and supports adaptive context-awareness. However, TOTA suffers from privacy issues and the network overhead which is incompatible with the fact that resources are scarce.

The contribution of this paper is the proposition of scoped tuples in a TOTA-based framework for the development of mobile ad hoc networking applications. This framework allows programmers to dynamically control the scope of tuples by preventing the transportation of tuples to devices which are not targeted. In the remainder of this paper, we describe in detail the TOTAM framework and the implementation of our case study in TOTAM. Before concluding, we outline the impact of our solution on the network traffic.

3 Scoped Tuples for the Ambient

Any TOTAM system is built from a number of *locations*. Every location x has a local tuple space T_x containing a set of tuples. Tuples hop from location to location in a wireless mobile ad hoc network by applying the TOTAM propagation protocol in every location. Tuples themselves carry behavior with them under the form of a number of operations. These operations are called on them in order to determine that they should be stored or propagated once they arrive at a certain location. To decide this the tuple has access to the local tuple space of the location where it arrives. Besides checking that the receiving tuple space is interested in a certain tuple at the receiving side, the TOTAM platform allows the tuple to check this before it is being physically transmitted. To be able to check this, each tuple space has a *tuple space descriptor*. This descriptor contains semantic information that can be used by the hopping tuples at *sending time* to decide whether a certain location is in their scope (by means of the `inScope` operation). For example, the descriptor for the ambient game application contains information about to which team the player belongs.

When two locations in the TOTAM network x and y move into each other's range, instead of immediately exchanging all tuples they first exchange their descriptors. Based on the received descriptor D_y, the location x sends a subset of its tuples T_x to location y. The subset of tuples $T_{x\to y}$ sent from x to y is defined as follows:

$$t \in T_{x\to y} \Leftrightarrow t \in T_x \wedge t.inScope(D_y) \tag{1}$$

Equation 1 defines that the tuples transfered from location x to location y which are exactly the subset of tuples t that are elements of the set of tuples of x with a scope extending to y. Checking that the scope of a tuple extends to a certain location y is done by applying the `inScope` operation of the tuple to the descriptor of the receiving tuple space, i.e. D_y.

Exchanging tuples according to this protocol allows the definition of flexible propagation strategies as the tuples themselves decide that they will be exchanged. This differentiates from techniques where the tuple space itself is scoped and tuples have to be inserted in a certain scope which can not be changed after its publication. By applying the concept of scope to the tuple, the tuple itself contains all the information to change it scope dynamically when hopping from location to location.

By disallowing the transportation of a tuple to a location which is not in its scope, potential *routers* of information are eliminated. Such routers carry tuples from one location to another without using them. This does not mean that routers are precluded by design in a TOTAM system. Within the scope of the tuple, TOTAM locations may still be used as routers. However, TOTAM locations forming part of the scope are considered as potential targets and the received information is not sensitive to them. For example, in the ambient game scenario, messages meant for a specific user in the blue team can be transported by other team members. But, such messages are never carried around by members of the red team which could use this information to their own advantage. Of course we cannot prevent malicious users in the scope of a tuple. This issue can be alleviated by combining scoped tuples with encryption techniques.

Figure 1 illustrates how a scoped tuple is propagated through the TOTAM network. It depicts two types of locations, the blue and red locations corresponding to the two teams of the ambient game scenario. The scope of the propagated tuple has been limited to blue locations. Figure 1(a) illustrates that a tuple is injected from the location with a star. This location is connected to four blue locations and one red location. As the scope of the tuple is limited to blue locations the tuple is only sent to the four blue locations. From those four locations the tuple is transitively propagated obeying the scope of the tuple until all connected blue locations are reached without

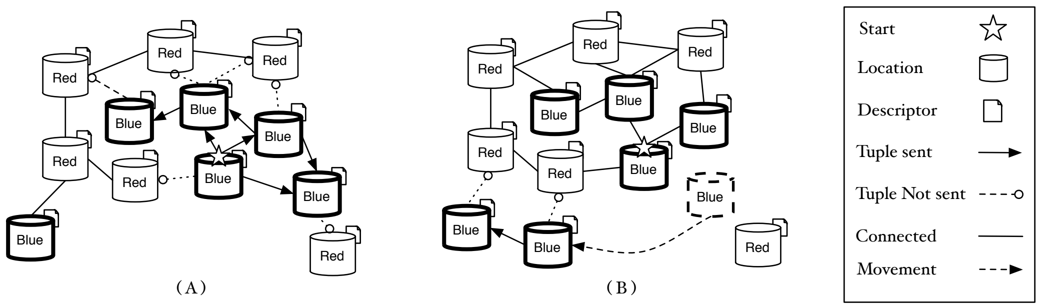

Figure 1: Operational sketch of a scoped tuple

being transmitted to a red location. Note that one blue location is not transitively connected to the sending device and thus does not receive the tuple. Figure 1(b) illustrates that a blue location moved into the range of the isolated blue location and thus, transmits the tuple to it. Again the tuple is not transmitted to nearby red locations. It is important to note from this operational sketch that the first isolated location receives a tuple without being connected at any time with the start location in which the tuple was originally inserted.

4 Programming in TOTAM

In this section we describe the set of operations provided by the TOTAM middleware to create tuple spaces, read and write from them and to define scoped tuples. TOTAM operations extend TOTA operations with the concept of scope by means of tuple space descriptors. Our middleware has been implemented on top of AmbientTalk, a distributed object-oriented programming language designed for mobile ad hoc networks [VMG+07]. TOTAM relies on an object-oriented tuple space and tuple representation. We introduce the necessary AmbientTalk syntax and features needed to understand our middleware while explaining the TOTAM operations.

4.1 TOTAM Primitives

In order to create a TOTAM location and add it to the network, programmers can call the `makeTupleSpace` operation as follows:

```
def TOTAM := makeTupleSpace(descriptor);
```

Variable definitions are defined with the keyword **`def`** and assigned to a value with the assignment symbol (:=). The operation `makeTupleSpace` takes a descriptor as parameter and initiates a new TOTAM location which exports itself to the *ambient*, i.e. the TOTAM network.

From then on the newly created location will look for other nodes in the TOTAM network and exchange its descriptor with them.The descriptor contains semantic information relevant to the propagation of tuples as explained later. The return value of this operation is an object which represents a newly created local tuple space which has a set of methods to interact with the middleware.

In order to insert tuples into the TOTAM network, the **inject:** operation can be called on the local tuple space as follows:

```
TOTAM.inject: tuple;
```

Once injected the tuple starts propagating in the TOTAM network according to the propagation rule (encapsulated in the tuple itself) and can be read and deleted from a location using the following two primitives.

```
TOTAM.read(template);
TOTAM.delete(template);
```

The `read` method defined on the local tuple space returns a collection (an array) of the tuples which matches the template passed by parameter. Reading does not remove the tuples from the tuple space and thus allows the tuples to be further propagated to other hosts. When this is not the wanted behavior the programmer should use the `delete` operation. This operation removes all tuples matching the template from the local tuple space and returns them to the caller in an array. In addition to these two operations, TOTAM provides an asynchronous operation **when:matches:** to register an observer on the local tuple space.

```
def subs := TOTAM.when: { |tuple| is: tuple taggedAs: Location } matches: { |tuple|
// update visual representation of the nearby player with the new position.
};
```

The **when:matches:** operation takes as argument a template and a closure which serves as an event handler. The handler will be called whenever a matching tuple is added to the local tuple space. In this example, an observer is placed on tuples transporting location information (checked by the **is:taggedAs:** function). The operation returns an object which can be used to cancel the matching subscription, by invoking `subs.cancel()`.

4.2 TOTAM tuples

The TOTAM framework defines a number of operations which have to be implemented by a tuple to control its scope and propagation. These operations with their default implementation are listed below and are provided to the programmer as the prototypical tuple.

```
def inScope(descriptor){ true };
def doAction(tupleSpace){};
def changeTupleContent(tupleSpace){ self };
def decideStore(tupleSpace){ true };
```

The operations of this prototypical tuple define a default propagation protocol (corresponding to a tuple which always propagates to every tuple space encountered as explained later). This prototypical tuple can be extended to encode custom propagation protocols by overriding the propagation methods. Once a tuple is created and inserted into the network it will be subject to the TOTAM protocol which calls the operations defined on the tuple in order to control its scope and propagation. The internal workings of this protocol are shown below.

```
// sending TOTAM location
if: tuple.inScope(descriptor){
  send(tuple, locationOf(descriptor));
};
```

```
//receiving TOTAM location
tuple.doAction(tupleSpace);
tuple := tuple.changeTupleContent(tupleSpace);
if: tuple.decideStore(tupleSpace) then: { tupleSpace.add(tuple)};
```

The `inScope` method is executed each time before transmitting a tuple to another location. It takes as parameter the descriptor of the receiving location and returns a boolean indicating whether it should be transported to that location. The default behavior of the `inScope` operation (defined by the prototypical tuple) returns `true`, the tuple is propagated to all connected locations.

At the receiver side, the first method executed upon tuple arrival (`doAction`) allows the tuple to perform some operations on the local tuple space (received by parameter) in which it has been propagated, i.e `read`, `delete` or **`inject`** tuples. A tuple can also add other tuples to the local tuple space by means of the `add` primitive. Successively, `changeTupleContent` is called on the tuple to update itself during the propagation process. The prototypical tuple doesn't change its content while it is propagated and thus this method just returns the tuple itself (i.e. **`self`**). Finally, the `decideStore` operation is called to determine whether the tuple has to be stored in the local tuple space. The prototypical tuple returns `true` and thus is stored in all the local tuple spaces of the locations where it arrives. Note that only tuples which store themselves in the local tuple store will be subject to propagation to other locations. The next section puts in practice the above tuple propagation protocol in the ambient game scenario introduced in section 2.1.

5 Implementation of the Ambient Game

This section describes how the TOTAM framework can be used to realise the ambient game scenario. We focus our discussion on the distributed coordination aspects of the game.

The ambient game application running on each player's PDA creates a TOTAM location which makes a player go online in the network. Every player is identified by a descriptor, called `myDescriptor`, encapsulating its username and the team that they belong to. Players periodically send their latest location information (e.g. GPS coordinates) to any nearby team member by means of location tuples. A location tuple can be encoded in TOTAM as follows:

```
def makeLocationTuple(username, location){
  extendTuple: makeTuple() with: { |myDescriptor|
    // propagation methods
    def inScope(descriptor) { descriptor.team == myDescriptor.team };
  } taggedAs: Location;
};
TOTAM.inject: makeLocationTuple(myDescriptor.username, [positionX, positionY]);
```

AmbientTalk objects are created by cloning and adapting existing prototypical objects, rather than created from a class. The `makeTuple` primitive returns the prototypical tuple object which is extended by means of the **`extendTuple:with:taggedAs:`** primitive. A location tuple has two fields (to store the username and the coordinates) and it is associated with the `Location` type tag (which we assume has been previously defined). Type tags are used in AmbientTalk to categorise objects explicitly by a nominal type. In this case, the type tag identifies the type of a tuple.

The player injects a new location tuple with the latest position (passed as argument). The tuple overrides the default propagation protocol to be only propagated to nearby team members.

In order to update the visual representation of nearby team members, for example a dot in the campus map, players subscribe to the arrival of location tuples as follows:

```
TOTAM.when: { |tuple| is: tuple taggedAs: Location } matches: { |tuple|
  // update visual representation of the nearby player with the new position.
};
```

when:matches: subscribes a players to tuples associated with the Location type tag. The **is:taggedAs:** function returns `true` if the object passed as parameter (`tuple`) is tagged as a given type tag (`Location`). In order for players to send each other messages to agree on a group strategy, a message tuple is modeled as follows:

```
def makeMessageTuple(from, content, to := nil){
  extendTuple: makeTuple() with: { |myDescriptor|
    def isTargeted(username){
      if: (to != nil) then: { to == username // message targeting a single team member
      } else: { true // message targeting all team members };
    };
    // propagation methods
    def inScope(descriptor){ descriptor.team == myDescriptor.team };
  } taggedAs: Message;
};
```

A message tuple has fields to store the content of the tuple (i.e. a text message) and the sender and the receiver of the text message. The receiver of the text message is initialized by default to **nil** (by means of the `to` optional parameter in the constructor function) which allows players to send messages to particular team member or to the whole team as follows:

```
TOTAM.inject: makeMessageTuple(myDescriptor.username,"policemen in front of building M");
TOTAM.inject: makeMessageTuple(
  myDescriptor.username, "policeman creeping up on you. Run!","tom");
```

A message tuple created without a particular receiver is sent to the whole team. With the `isTargeted` method players test whether the message targets the entire team or a particular team member. Note that team members can be used as routers to transport messages to a particular player of their team. This behaviour is useful in the ambient game in order for players to receive messages from a particular team member even if they were never connected at the same time. However, as the content may not be interesting for the routers, players may just subscribe to messages targeted to them directly or to all team members as follows:

```
TOTAM.when: { |tuple| (is: tuple taggedAs: Message) &&
  tuple.isTargeted(myDescriptor.username) } matches: { |tuple|
  // display text message on the gui
};
```

Finally, a player can drop a bomb (tuple) that affects players of the opposite team in a certain range (defined by the numbers of hops from the player that drop it):

```
def makeBombTuple(range){
  extendTuple: makeTuple() with: { |myDescriptor|
    def inScope(descriptor) { (descriptor.team != myDescriptor.team) && (range > 0)};
    def doAction(ts) { // notify the player that she/he has been bombed! }
    def changeTupleContent(ts) { range := range - 1; self};
  } taggedAs: Bomb;
};
```

A bomb tuple has a field (`range`) to store the number hops it can perform. `inScope` returns `true` when the receiver tuple space is a member of the opposite team and the tuple has not been already propagated to all tuple spaces reachable in n-hops. `doAction` notifies the reached players that they were bombed. A more advanced implementation of the bomb tuple could affect a player by e.g. decreasing his/her life total. `changeTupleContent` updates the tuple being propagated by decreasing the `range` value by one at every hop.

6 Discussion and Evaluation of network traffic

In this section, we evaluate TOTAM in view of the issues identified in Section 2. As TOTAM adopts a tuple-based model inspired by the TOTA middleware, our approach inherits the solutions for the discovery, context-awareness and communication issues: (1) TOTAM locations can discover each other spontaneously, (2) the propagation protocol equipped in tuples cater to context-awareness in an adaptive way, and (3) TOTAM locations do not have to be available in the network at the same time to exchange tuples.

TOTAM addresses the privacy and scarce resources issues by introducing the notion of scope on tuples. By making use of tuple space descriptors, programmers can scope their tuples preventing them to be transported to unwanted locations. These descriptors are crucial to provide programmers with a hook to encode privacy strategies. Although the descriptors in the ambient game are limited to be simple objects carrying the team identifier this can be extended to compute an encryption challenge. By avoiding the unnecessary tuple transportation, our approach can minimize network traffic. Tuple space descriptors are exchanged between two locations when they meet for the first time and whenever a location decides to change its description. In case descriptors stay constant and prevent the propagation of tuples they can drastically reduce the burden on the network. In the other case when descriptors change a lot or do not prevent the transportation of tuples the danger exists that the network traffic gets dominated by the transmission of descriptors. In the remainder section we evaluate when the use of tuple space descriptors is beneficial and in which cases it is not in terms of network traffic.

6.1 Worst Case

In the worst case there is one message that has to be transported to all connected locations. This means that the exchange of the tuple space descriptors is an overhead as the tuple was unlimited in its scope i.e. the tuple floods the network. The network traffic generated for this tuple to be sent over the network when two locations meet can be computed as follows. Every location x connecting to a location y will first receive the descriptor D_y over the network and then receive the tuple t_{x1}. The total amount of network traffic for this tuple can thus be computed by summing the exchange of all descriptors with the total amount of exchanged tuples. This is shown in the following equation where n represents the number of connected locations.

$$NetworkTraffic = (\sum_{x=0}^{n}\sum_{y=0}^{n} D_y) + n.t_{x1} \tag{2}$$

In case the descriptors do not change they will only be transported once when two locations discover each other. This means that from the second communicated tuple the cost of the descriptor is dropped in the above equation. The resulting equation is exactly the traffic that is normally transferred ($n.T_{x1}$) when not making use of tuple space descriptors. However in the worst case all connected locations change their descriptors for every transmitted tuple. In this case the overhead of transferring the descriptors is given by the following equation where N is the number of transferred tuples.

$$NetworkOverhead = (\sum_{x=0}^{n}\sum_{y=0}^{n} D_y) * N \tag{3}$$

The overhead of exchanging the descriptors will be quadratic to the number of connected locations over time. This clearly shows that when descriptors change a lot and tuples have to be sent to all connected locations encountered it is not beneficial to use tuple space descriptors.

6.2 Best Case

In the best meaningful case[1] the sent tuples are sculpted to be only sent to one location and the tuple space descriptors do not change over time. We illustrate this case by a tuple that hops from location to location in a ring. In every hop the tuple adjusts its scope to the next hop in the ring configuration. We have illustrated the network traffic generated by this scenario for TOTAM and traditional approaches in figure 2. It is important to split up the network traffic generated by TOTAM in the case for the first tuple and the successive ones. As can be observed on the top left of the figure, before sending the first tuple over the ring all descriptors have to be exchanged. However, when this tuple is further propagated over the ring exchanging the descriptors is not necessary anymore so the number of exchanged tuples for one round equals the size of the ring (as shown in the second and third step of the figure). This is in contrast to approaches where the tuples are not scoped, in these cases the number of exchanged tuples for one round is quadratic to the number of locations participating in the ring.

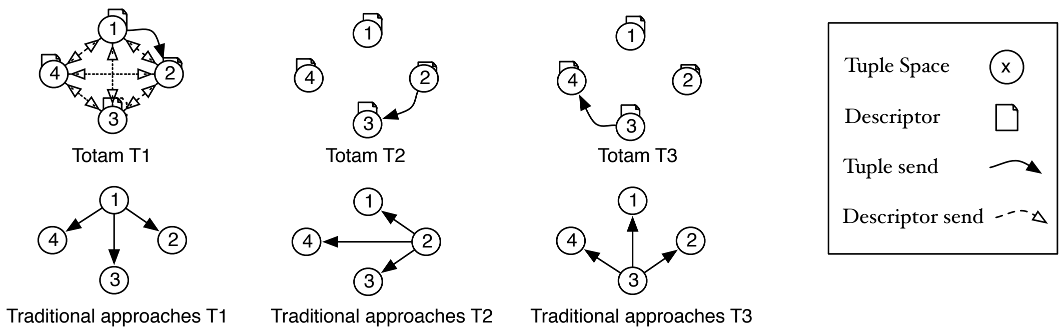
Totam T1
Totam T2
Totam T3
Traditional approaches T1
Traditional approaches T2
Traditional approaches T3
Tuple Space
Descriptor
Tuple send
Descriptor send
x
1
2
3
4

Figure 2: Set-up of best case scenario

[1] The theoretical best case is when the tuple is meant for nobody and thus does not generate network traffic at all.

After evaluating the worst and the best case it is clear that the use of tuple space descriptors in combination with scoped tuples has the potential to drastically reduce the network traffic when 1) the tuples will be prevented from hopping to other locations and 2) the descriptors do not change often relative to the number of tuples in the system.

7 Related Work

A number of approaches support scoping mechanisms in the context of tuple spaces. Coordination with Scopes [MW00] introduces the concepts of scope for a tuple space. A scope represents a view on a flat tuple space. A set of operations is defined on those views allowing scopes to be joined, nested, intersected and subtracted. Tuples may thus be visible from several different scopes. This mechanism is mainly used to structure tuple spaces according to different viewpoints on a flat tuple space. However, they do not limit the propagation of tuples, i.e. the tuple is propagated to other tuple spaces but it may not be visible for certain scopes. Since the system was not devised for mobile computing applications, they rely on centralized infrastructure. In contrast, TOTAM does not rely on any fixed infrastructure and tuples can be propagated through spontaneously formed mobile ad hoc networks.

CAMA[IA06] is an agent-based tuple space system which allows the definition of a scope which agents can join and leave. Scopes are defined as containers in a tuple space and can be nested to form hierarchical structures. This notion of scope improves on coordination with scopes since inserted tuples are only transmitted to agents which reside in the same scope. However, in order to send tuples to other scopes the agent first needs to change its scope. Tuples which are inserted in a specific scope can not be propagated automatically to other scopes. In TOTAM, by allowing the tuple itself to decide whether it should be propagated, more fine-grained sharing strategies can be expressed.

L2imbo is a tuple-space based platform for mobile computing which provides special features for quality of service. Similar to CAMA, L2imbo introduces the concept of multiple tuple spaces but suffers from the same limitations as tuples do not have the ability to change to which tuple space they should be propagated. It is interesting to note that L2imbo supports time-outs to be associated with tuples which makes possible for the system to reorder tuples to make optimal use of the available network connectivity. This behaviour can be achieved in TOTAM by manually encoding it in the tuple propagation protocol.

Evolving tuples [SJ07] have a field destination that they can change while they are hopping. This destination field is used to determine where the tuple will be transmitted to after leaving a host. However, the destination field can only be a broadcast address or a specific host address. In order to send the tuple to two broadcast addresses the programmer will have to read the tuple and reinsert it to another broadcast address. Our approach uses semantic information to determine where it can be transmitted thus allowing more fine-grained propagation rules.

Inspired by LIME, TeenyLIME [CMMP06] introduces abstractions specially designed for wireless sensor networks (WSN). Every tuple space in TeenyLIME is shared only with one-hop neighbours, limiting the scope of tuples to one hop. Such limitation fits natural with WSN architectures where every node typically needs access to nearby information [CMMP06]. Scoping was introduced to address the scarce resources issue in the context of WSN. However, no other means are provided to express different propagation protocols, which need to be expressed in terms of single-hop operations.

Other loosely decoupled communication models like publish/subscribe systems have explored the concept of scope. In publish/subscribe systems, subscribers register for events and are *asynchronous* notified when a publisher generates a matching event. Matching is usually performed by an event broker which publishers post events to and subscribers register themselves with. In larger systems this event broker is extended to be an acyclic connected graph of event brokers, such as in REBECA [M02]. In this discussion we do not consider systems which use such a graph of fixed event brokers as it does not scale for mobile networks. In scoped REBECA [LMMB02] systems can create a scope in which events will be published. A scope can be extended and thus form a tree of scopes. Subscribers will only receive events of publishers which are in the same scope or have a common ancestor in the scope hierarchy. Similar to CAMA, publishing an event in an other scope requires the publisher to change it scope first. STEAM [MC03] allows publish/subscribe based on physically location, but it is hard to describe scopes based on semantic information as shown in the ambient game. Location-based publish/subscribe [EGH05] suffers from the same limitation. Frey et al. [FR07] allow the limitation of publications and subscriptions on both physical and semantic information (from the publishing and receiving nodes). However, events in their system cannot adapt themselves while being propagated.

8 Conclusion and Future Work

We have introduced a novel tuple space-based approach which provides a dynamic scoping mechanism that limits the transportation of tuples. By means of tuple space descriptors, programmers can scope their tuples preventing them to be propagated to unwanted locations. These descriptors are exchanged between two locations when they meet for the first time and whenever a location decides to change its description. The novelty of our approach lies in the use of these tuple space descriptors to determine the scope of tuples *before* they are being transmitted. This enhances privacy and decreases the burden on the network traffic in a wide range of applications.

We would like to extend the `doAction` propagation operation with a guard-like predicate that is periodically evaluated until it allows the `doAction` to be triggered. This is different from performing the same predicate check in the `doAction` as this operation is evaluated only once when the tuple arrives at its new location. We are currently developing a toolkit for ambient applications called UrbiFlock on top of TOTAM. Urbiflock is a Facebook-like application framework specially designed to enable spontaneous interaction of people in the campus of the Vrije Universiteit Brussels by means of mobile devices (such as their cell phones). We believe that the implementation of UrbiFlock will help to identify possible shortcomings and points to improve.

Acknowledgements: Christophe Scholliers is funded by a doctoral scholarship of the Institute for the Promotion through Science and Technology in Flanders (IWT-Vlaanderen). Elisa Gonzalez Boix is funded by the Prospective Research for Brussels program of the Institute for the encouragement of Scientific Research and Innovation of Brussels (IWOIB-IRSIB). The authors would like to thank Tom Van Cutsem, Stijn Mostinckx and the anonymous reviewers for their helpful comments and suggestions for improvement.

Bibliography

[CMMP06] P. Costa, L. Mottola, A. Murphy, G. Picco. TeenyLIME: transiently shared tuple space middleware for wireless sensor networks. In *MidSens '06: Proceedings of the international workshop on Middleware for sensor networks*. Pp. 43–48. ACM, New York, NY, USA, 2006.

[DFWB98] N. Davies, A. Friday, S. Wade, G. Blair. L2imbo: a distributed systems platform for mobile computing. *Mob. Netw. Appl.* 3(2):143–156, 1998.

[EGH05] P. Eugster, B. Garbinato, A. Holzer. Location-based Publish/Subscribe. *Fourth IEEE International Symposium on Network Computing and Applications*, pp. 279–282, 2005.

[FR07] D. Frey, G. Roman. Context-Aware Publish Subscribe in Mobile Ad Hoc Networks. In *9th International Conference on Coordination Models and Languages (COORDINATION)*. Lecture Notes in Computer Science 4467, pp. 37–55. Springer-Verlag, June 2007.

[Gel85] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems* 7(1):80–112, Jan 1985.

[IA06] A. Iliasov, R. A. Structured coordination spaces for fault tolerant mobile agents. *Advanced Topics in Exception Handling Techniques* 4119:181–199, 2006.

[LMMB02] F. Ludger, M. Mezini, G. Mühl, A. Buchmann. Engineering Event-Based Systems with Scopes. In *ECOOP '02: Proceedings of the 16th European Conference on Object-Oriented Programming*. Pp. 309–333. Springer-Verlag, London, UK, 2002.

[MÖ2] G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, 2002.

[MC03] R. Meier, V. Cahill. Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications. In *Proceedings of the 4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'03)*. 2003.

[MCE02] C. Mascolo, L. Capra, W. Emmerich. Mobile Computing Middleware. In *Advanced lectures on networking*. Pp. 20–58. Springer-Verlag New York, Inc., 2002.

[MPR01] A. Murphy, G. Picco, G.-C. Roman. LIME: A Middleware for Physical and Logical Mobility. In *Proceedings of the The 21st International Conference on Distributed Computing Systems*. Pp. 524–536. IEEE Computer Society, 2001.

[MW00] I. Merrick, A. Wood. Coordination with scopes. In *SAC '00: Proceedings of the 2000 ACM symposium on Applied computing*. Pp. 210–217. ACM, New York, NY, USA, 2000.

[MZ04] M. Mamei, F. Zambonelli. Programming Pervasive and Mobile Computing Applications with the TOTA Middleware. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PERCOM)*. P. 263. 2004.

[SJ07] D. Stovall, C. Julien. Resource discovery with evolving tuples. In *ESSPE '07: International workshop on Engineering of software services for pervasive environments*. Pp. 1–10. ACM, New York, NY, USA, 2007.

[VMG$^+$07] T. Van Cutsem, S. Mostinckx, E. Gonzalez Boix, J. Dedecker, W. De Meuter. AmbientTalk: object-oriented event-driven programming in Mobile Ad hoc Networks. In *Proceedings of the XXVI International Conference of the Chilean Computer Science Society (SCCC 2007)*. Pp. 3–12. IEEE Computer Society, 2007.