TripCom

Triple Space Communication

FP6 – 027324

Deliverable

D6.5

Towards a Scalable Triple Space

Reto Krummenacher
Elena Simperl
doug foxvog
Vassil Momtchev
Dario Cerizza
Lyndon Nixon
Davide Cerri
Brahmananda Sapkota
Kia Teymourian
Philipp Obermeier
Daniel Martin
Hans Moritsch
Omair Shafiq
David de Francisco

Executive Summary

The TripCom scalability task was defined as reaction to the feedback received during the first project review. This new effort is part of WP6 and is dedicated to the study of core principles at design, architecture and implementation level in order to ensure the realization of a Web-scale Triple Space.

The roadmap behind this deliverable is divided into six steps: define the overall objective of the task, ii) specify the core concepts and definitions regarding scalability and its specific meaning to our project, iii) identify scalability factors and their mutual influences, iv) introduce so-called *configurations*, v) define and discuss the design of the deployment architecture of Triple Space, and vi) present a procedure for the evaluation of our system in a large, distributed environment.

Document Information

IST Project Number	FP6 – 027324	**Acronym**	TripCom
Full Title	Triple Space Communication		
Project URL	http://www.tripcom.org/		
Document URL			
EU Project Officer	Werner Janusch		

Deliverable	**Number**	6.5	**Title**	Towards a Scalable Triple Space
Work Package	**Number**	6	**Title**	Triple Space Architecture and Component Integration

Date of Delivery	**Contractual**	M24	**Actual**	31-March-08
Status	draft		final ☒	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Reto Krummenacher			
Resp. Author	Reto Krummenacher		**E-mail**	reto.krummenacher@sti2.at
	Partner	Semantic Technology Institute, University Innsbruck)	**Phone**	+43 512 507 6452

Abstract (for dissemination)	This deliverable is dedicated to the study of core principles at design, architecture and implementation level in order to ensure the realization of a Web-scale Triple Space.
Keywords	Scalability, configurations, evaluation, Web-scale Triple Space

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
2007-08-06	1	Elena Simperl	Preliminary structure and content outline
2007-08-16	2	Elena Simperl	Revised structure and content outline
2008-01-27	3	Vassil Momtchev	Triple Space Persistent Storage section added
2008-01-31	4	Brahmananda Sapkota	Deployment architecture section added
2008-02-02	5	Kia Teymourian	Distribution Strategy added
2008-02-08	6	Brahmananda Sapkota	Updated section 5.1
2008-01-25	7	Daniel Martin	Added Section 6.1
2008-02-11	8	Omair Shafiq	Input added in section 6.1
2008-02-18	9	David de Francisco	Input added in section 6.3
2008-02-28	10	Brahmananda Sapkota	Updated section 5.1
2008-02-28	11	Daniel Martin	Drafted Conclusions
2008-02-29	12	Omair Shafiq	Updated section 6.1
2008-03-08	13	Kia Teymourian	Updated section 5.1
2008-03-10	14	Brahmananda Sapkota	Updated section 5.1

Project Consortium Information

Acronym	Partner	Contact
Semantic Technology Institute Innsbruck http://www.sti-innsbruck.at	STI	Prof. Dr. Dieter Fensel Semantic Technology Institute (STI) Innsbruck, Austria E-mail: dieter.fensel@sti-innsbruck.at
National University of Ireland, Galway http://www.deri.ie	NUIG	Dr. Laurentiu Vasiliu Digital Enterprise Research Institute (DERI) Galway, Ireland Email: laurentiu.vasiliu@deri.org
University of Stuttgart http://www.iaas.uni-stuttgart.de/	USTUTT	Prof.Dr. Frank Leymann Inst. für Architektur von Anwendungssystemen (IAAS) Stuttgart, Germany E-mail: frank.leymann@informatik.uni-stuttgart.de
Vienna university of Technology http://www.complang.tuwien.ac.at/	TUW	Prof.Dr. eva Kühn Institut für Computersprachen Vienna, Austria E-mail: eva@complang.tuwien.ac.at
Free University Berlin http://www.ag-nbi.de/	FUB	Prof. Dr.-Ing. Robert Tolksdorf AG Netzbasierte Informationssysteme Berlin, Germany E-mail : tolk@inf.fu-berlin.de
Ontotext Lab, Sirma Group Corp. http://www.ontotext.com/	ONTO	Atanas Kiryakov, Vassil Momtchev, Ontotext Lab, Sirma Group Corp. Sofia, Bulgaria E-mail: vassil.momtchev@ontotext.com
Profium OY http://www.profium.com/	Profium	Dr. Janne Saarela Profium OY Espoo, Finland E-mail: janne.saarela@profium.com
CEFRIEL SCRL. http://www.cefriel.it/	CEFRIEL	Davide Cerri CEFRIEL SCRL. Milano, Italy E-mail: cerri@cefriel.it
Telefonica I+D http://www.tid.es/	TID	Noelia Pérez Crespo Telefonica I+D Madrid, España E-mail: npc@tid.es

Table of Contents

List of Abbreviations

AMI	Amazon Machine Image
API	Application Programming Interface
CAN	Content-Addressable Network
CMR	Complete-Model-then-Revise
DAM	Digital Asset Management
DBMS	Database Management System
DHT	Distributed Hash Table
DNS	Domain Name Server
EAI	Enterprize Application Integration
EC2	Elastic Compute Cloud
EPS	European Patient Summary
EU	European Union
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
OWL	Web Ontology Language
P2P	Peer-To-Peer
RDF	Resource Description Framework
RDFS	RDF Schema
SSH	Secure Shell
SPARQL	SPARQL Protocol and RDF Query Language
S3	Simple Storage Service
TCP	Transmission Control Protocol
TripCom	Triple Space Communication
TS	Triple Space
TUW	Technische Universität Wien
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WP	Work Package
WS	Web Service
WSMO	Web Service Modeling Ontology
WSMX	Web Service Execution Environment
XML	Extensible Mark-up Language

1 Introduction

The TripCom scalability task was launched in response to the recommendations of the project reviewers. This scalability study is dedicated to the core principles at design, architecture and implementation level in order to ensure the realization of the Web-scale Triple Space. The goal of the task is to determine the relevant factors, to analyze the potentials of various applicable technologies and approaches, and to present scalable solutions to Triple Space. Moreover, it is the aim of this deliverable - at least in its second release - to provide analysis tools and evaluations to determine the level of scalability of an implementation and to give proof of its application at Web scale.

The scalability task consists of the following steps, which are carried on sequentially:

1. **Define overall objectives** The overall objectives of the scalability task are defined in accordance with the description of work.

2. **Specify core concepts and definitions** Describe the TripCom view of scalability in detail and sketch the general approach, as well as the procedures to evaluate the scalability of the Triple Space platform.

3. **Identify scalability factors, assumptions, trade-offs** Provide a complete description of the scalability factors and the associated trade-offs or mutual relations that influence the scalability of the system. This step is central for the scalability task.

4. **Develop and analyze configurations** Next, the factors are taken over as parameters for the development of configurations and analysis of Triple Space implementations. The configurations provide a formalized means to describe a particular triplespace realization, while the analysis delivers a means to describe the scalability level and scalability bottlenecks of the realization in terms of functional and non-functional properties.

5. **Realize Triple Space** Use the results of the Steps 3 and 4 to realize a scalable Triple Space. For this purpose we revise the core components of the middleware, as well as the deployment architecture and show how they align with various configurations, i.e. with different levels of functionality and scalability.

6. **Evaluate results for scalability** Evaluate the final system with respect to its scalability, based on pre-defined procedure and concepts (cf. Step 2).

The aforementioned steps are one by one addressed in this deliverable. The first bullet is subject to this introductory section. The subsequent steps 2-4 are considered in the respective Sections 2-4. First, TripCom's view on scalability is defined (Section 2). Section 3 introduces the core notions of functional and non-functional properties, their influences on scalability and trade-offs in order to come up with a set of relevant parameters (Step 3) for the evaluation of Triple Space; these parameters become parts of the triplespace configurations. These configurations are on the one hand a description of a particular realization, while on the other they serve as input to the scalability analysis task (Section 4). In Section 5 and Section 6 this deliverable discusses the choices taken for the TripCom Triple Space implementation, and

the resulting impact on the use case realizations. These sections correspond to step 5 above. Lastly, in Section 7, methodologies, test suites and results are presented that allow the evaluation of the concrete Triple Space prototype (Step 6).

2 Scalability in Triple Space

Scalability is a desirable property of a process, a network, or a (e.g. database) system, which indicates its ability to either handle growing amounts of work in a graceful manner, to be readily enlarged to meet additional demand, or both. A system which can balance an increase in demand with a proportional increase in hardware and whose performance improves after adding hardware, proportionally to the capacity added (up to some limit), is said to be a linearly scalable system.

For algorithms and protocols, scalability is typically defined by complexity measures ($O()$, etc.). Whether a system is considered to be scalable or not depends to a large extent on the state-of-the-art of a specific domain. Typical highly scalable algorithms in many domains are of complexity $O(log(n))$.

Equations defining cost and performance measures of scalability are problematic in the general case and not suitable for Triple Space. Such equations often consider each attribute of an application to be independent of all but one or two of the others, not taking into account any interactions or trade-offs among them. For such reasons, the TripCom definition of scalability is not equation based.

A fully scalable database can be expanded with a graceful effect on the data access time. By analogy a tuplespace can be treated as a distributed database system for scalability purposes. Similar to a database system, the performance of a scalable tuplespace network would increase gradually as additional spaces and kernels are added (assuming that additional resources are added as needed).

For Triple Space, scalability measures would include data access time (latency) for its core functionalities (reads, writes, deletions, and notifications), frequency and parallelism of such requests, triplespace size, number of spaces, number of processors over which a space is distributed, number of users, and the number of internal messages generated to route a request from a Triple Space entry point to a machine which hosts the required subspace.

If one of these measures is increased while most of the rest are held fixed, the other(s) should increase in a graceful way. However, interactions among them can be quite complex.

Latency is a key scalability measure. Although an *out* returns immediately, not waiting for the output to be stored, and *rd*, *rdmultiple*, *in*, and *inmultiple* allow the specification of timeouts and the $-multiple$ versions do not guarantee a complete set of matches, data access times of days or longer do not make for a functioning system. Latency scaling must be logarithmic for measures that may expand exponentially (or at least many orders of magnitude over test cases). If the latency for a use case using 1,000 records in 1,000 subspaces is one second, and it needs to scale up to 400 million subspaces – the size of a full European Patient Summary system – linear scaling of data access time with respect to number of spaces would mean the access could take over four days per operation, while with $O(log(n))$ scaling, the operation might take under 20 seconds.

Increased frequency and parallelism of requests require a proportional increase in the bandwidth of the system once initial bandwidth limitations start degrading performance. Bandwidth may be increased by increasing the bandwidth of individual machines (up to a point), by distributing the triplespace among machines, and/or by replication of data on multiple machines.

As a database increases in size, proportionally more computer memory must be made available to store it. The increase may be on an individual machine up to a point, but must eventually be handled by distribution of the memory among networked machines.

Similarly, as triplespaces are distributed over multiple machines, additional internal messaging, storage, and processing is required. Redundancy becomes necessary. The amount of redundancy necessary could be affected by the number and geographical distribution of both memory devices and users, network reliability and demand, and criticality of data, among other issues. Data in a single triplespace may be distributed into multiple subspaces to make the distributed data easier to handle.

The number of authorized users should have a minimal effect on the system. Millions of health care workers would have access to a European Patient Summary, even if the hundreds of millions of patients were not provided read access to their own records.

Some scalability comparisons may be ternary, such as: "at what rate need number of subspaces scale with respect to total triplespace size if latency is to scale at $O(log(n))$?"

Determining what scalability complexity classes should be considered "good" or acceptable is difficult without knowing the complete system. What is good or acceptable for one use case may not be so for another. However, some example scalability rules of thumb for Triple Space might be:

- Data transmission time should scale no worse than linearly with respect to the amount of data transmitted in an operation – we do not intend to implement adaptive data compression schemes which could yield sublinear scaling.
- Data access time should be highly scalable with respect to the number of computers in the network, although the number of computers need only scale linearly with respect to the number of kernels and subspaces – orders of magnitude more computers would be used to store patient data in a full-scale EPS than in a prototype testbed.

For a system which does not guarantee 100% accurate and consistent results, properties of the triplespace which measure aspects of result variability – such as availability, reliability, fault-tolerance, security, safety, completeness, correctness, consistency, and durability – and trade-offs among them should be addressed with their effects on scalability in mind. These issues are each explored in more depth in the next chapter.

Although scalability in general is not equation based, quantitative equations for measuring the order of scalability of one parameter against another under constrained circumstances allow for evaluation methods based on performance measures such as response time (cf. Section 7.1). Aspects of scalability captured this way represent a relationship among resources, demands on the system and performance measures in restricted circumstances. Care must be taken in generalizing the results beyond the range of the set of conditions which have been tested. An attribute which had no affect on the measurements being tested within the range of the test, e.g. communication line maximum bandwidth, might become a limiting factor after further scaling of the system. Judging the limits for which a derived equation is valid is not an easy task.

Given the above considerations, scalability for Triple Space can be defined as a property which allows a triplespace system to maintain acceptable performance by expanding in a graceful way by adding components to handle increasing demand for memory, bandwidth, operations, and users.

3 Scalability Factors

The scalability factors are those characteristics of a distributed system that influence the potential scalability limits of a particular implementation. There are two types of factors, which we refer to as functional, respectively non-functional properties. The former describe the exposed functionality of the Triple Space, i.e. the service that is delivered to and perceived by the clients. The latter are then the aspects that describe the internal behavior of the system, i.e. the properties that are not obvious or visible to clients at interaction time. Such non-functional aspects are for example availability, reliability, fault-tolerance, completeness, correctness, durability, response time, and also security.

3.1 The Functional Aspects

The functional description, i.e. the core functionality of Triple Space can be described as follows:

Publication: An application can publish information to a space for which it has the required write access rights. This information is eventually available in the space for further processing and persistently stored. This corresponds to an `out` operation in terms of the Triple Space API.

Retrieval: An application can read information from a space for which it has read access. The information can be read multiple times, as defined by the operation ***rd*** in the Triple Space API (cf. Section 5.2 or [25]. The retrieval is governed by template matching algorithms for which the complexity depends on the expressivity of the syntax applied: simple triple patterns, graph patterns or SPARQL queries. Retrieval can moreover be performed on a particular triplespace, or against the whole Triple Space.

Removal: Information can be removed, or invalidated from a space by any client that possesses the required access rights. Removal of an information item has the effect that this item is no more available as truth value in the space for further readings; and hence, removal can be installed as a variant of publish and retrieve through the publication of invalidating statements.

Notification: Clients can subscribe to particular types of information and are then notified whenever the requested information becomes available. Subscriptions are also governed by templates, and hence similar conditions apply as for regular retrieval. In other words notification too, is only a variant of retrieve, however with distinct operations in the Triple Space API.

Orthogonally to the presented functionalities we need to differentiate between a data-driven and a knowledge-driven Triple Space. The former is assumed to manage RDF (or any other semantic data) at syntactical level. By contrast the latter takes the interpretation of the data into account by the given machine-understandable semantics. This results in particular in the fact that operations can be carried out not only on asserted, but also on inferred data generated using specific reasoning engines. Further aspects which impact the functional level are thus:

Explicit Data vs. Inferred Data (Knowledge): Operations can take into account the semantics of the published data or just its syntax. Publication at data level must permit the availability of multiple copies of the same triple, whilst at knowledge level, one statement (truth value) exists only once. Consider for example invalidation of published information: at data level every triple needs to be handled separately, while at knowledge level invalidation concerns all related statements. This complicates the task, as at knowledge level the middleware must ensure that all distributed copies are invalidated. Furthermore, all operations have to specify to which extent they are carried out on asserted or inferred data. In terms of publication this means that the availability of a new item could cause new inferences to be processed and new facts to be materialized. A retrieval operation will then ask for inferred information to be returned as a result just as explicitly available data.

Query Complexity: Triple Space data supports templates/queries with various degrees of complexity. In its easiest form, as stated above, data querying is limited to single triple patterns. Such matching approaches can scalably be realized for large open systems like existing semantic P2P-implementations have shown [2, 4].[1] As the target applications of Triple Space demand more complex query languages, or even rule-based query resolution; e.g. [19]. These more complex matching algorithms provide more functionality at the price of a more complex implementation (Section 5.2).

3.2 The Non-Functional Aspects

In this section we present the non-functional properties of middleware. Besides a short explanation of the non-functional properties we outline a number of measures that are necessary, not implicitly sufficient to realize a Triple Space that conforms to that non-functionality.[2] We first introduce the four measures that Triple Space has at its disposal: redundancy of Triple Space operability, recovery of data and operability, load balancing either by use of distribution or replication, and synchronization.

Redundancy: Redundancy refers to the principle of having additional or duplicate systems, equipment, etc., that function in case an operating part or system fails. First, there might be redundant components within a Triple Space (TS) kernel that ensure the expected functionality. More importantly however, redundancy in our context refers to the fact that multiple kernels might deliver the same service to users in case a primary access point fails or is temporarily not available.

Recovery: In case of failure recovery enables the reinstallation of a prior operational state, in particular also with respect to the published and stored data. There are two types of recovery recognized in computer science: roll-forward or roll-back. The former takes the system state at that time and corrects it, to be able to move forward, while the latter reverts the system state back to some earlier, correct version, for example using checkpointing, and moves forward from there.

[1] Simple triple patterns are resolved in $\log(N)$ with N the number of nodes.

[2] Under the assumptions discussed in the next section: openness, dynamism, distribution, decentralization, and scalability.

In Triple Space, recovery depends a lot on the persistency framework of the TS kernels that stores internal management data of the various components.

Load Balancing: Redundancy uses duplicate kernels without increasing the available access points. Load balancing relies on kernels with equivalent execution semantics that run in parallel. Load balancing is a means to distribute responsibilities and requests and is at least partially based on distribution or replication of data:

Replication: The replication technique clones the published data and stores it at different locations within the Triple Space network for the case that the primary source is not or no longer accessible.

Distribution: As opposed to replication, distribution of data does not copy data, but rather partitions and distributes the data across multiple kernels.

In case load balancing is based on replication the service delivered is expected to be the same for all kernels, while with data distribution the service is potentially different, but still semantically correct; i.e. the data retrieved from different kernels might differ, as complete access to all data cannot be guaranteed. This does however not violate the definition of Triple Space retrieval.

Synchronization: This technique refers to the procedures that allow the coordination of actions and processes. In the sense of this work, it also refers to the timely adjustment of data. In other words, synchronization is applied to keep copies of the same data or the same process equipollent, and hence in a state of consistency.

Having the available techniques in place we now present the non-functional properties and depict how they relate to Triple Space.

Availability: This property describes the total time a computing system is up and running, and hence the ratio or probability that the system is accessible and providing the promised service. Numerically the availability is given by the ratio of the expected value of the uptime of a system to the aggregate of the expected values of up and down time. High availability is thus the general goal, in order to guarantee a certain degree of operational continuity.

Technical measures: Load balancing, i.e. the distribution of responsibilities, is a very promising measure to increase the operational continuity of a system. Care must however be given to the system architecture, as decentralization often carries along a more complex installation, which in turn is more vulnerable to system failures.

Role and Relevance in Triple Space: Availability in Triple Space is first of all dependent on the accessibility of TS kernels and hence inherits primarily the availability of the underlying implementation. However, in the case of a distributed space, i.e. the case where a space is co-authorized by multiple kernels, availability depends not only on the kernel hosting the space, but only on the availability of one of the sharing kernels. A space is accessible, also if only partly, as soon as one of the kernels hosting it can be reached. In that case completeness is however not guaranteed, while higher availability can be ensured.

Reliability: Reliability is known as the ability of a system or component to perform its required functions under stated conditions for a specified period of time.

Technical measures: A measure against a lack of reliability is redundancy of the internal processes that control each other in a two-out-of-two partnership. The problem with replication is the possible loss of consistency, as it is not ensured that multiple copies of the same resource reflect the same state and content.

Role and Relevance in Triple Space: Reliability in Triple Space is measured against the expected functionality and non-functionality of a configuration (cf. Section 4.1 for the definition of configurations). There is no default solution for the installation of a reliable Triple Space, but many configuration-dependent ones. As example, if availability is not a concern at all (i.e. either we get an answer or not), but completeness is highly relevant (i.e. if we get an answer then we get all the triples), the expected service of the Triple Space middleware is only measured against the qualitative performance of the completeness property.

Fault-tolerance: Fault-tolerance allows a system to continue operating properly in the event of a failure of one or more of its components. In other words, the predominant factor with respect to fault-tolerance is the ability to recover.

Better is however to avoid system failures per se. This is first of all achieved by keeping the system complexity low, and secondly through redundancy of internal components and processes. Analogous to the two-out-of-two approach mentioned for reliability, systems with significant fault-tolerance requirements rely on so-called two-out-of-three solutions. In such settings the component or process detecting a failure has time to recover without negatively influencing the overall performance of the system.

Technical measures: The ability to recover is an important factor with respect to fault-tolerance. However, more important are the following measures that help to avoid system failures:

- No single point of failure.
- No single point of repair (the system must run during the repair process).
- Fault isolation to prevent propagation of the failure.
- Availability of recovery procedures.

Role and Relevance in Triple Space: Fault-tolerance primarily depends on its counterpart of TS kernels and their components. Additionally the accessibility of a kernel, i.e. the failure tolerance of the network, influences the resistance to errors. Here again, Triple Space could install redundant or parallel kernels that share a single space, which increases the probability of failure-free communication. Moreover, this ensure enough time for erroneous kernels, components and network links to be reinstalled and recovered.

Security: This non-functional property refers to the condition of being protected against danger or loss, and defines how the middleware is protected against malicious agents attempting to cause failure or even destruction. Security is a very important feature also with respect to the aforementioned properties, as

negative influences from the outside heavily influence the availability, reliability and fault-tolerance of a system.

Security is a very good example of non-functionality: the service that a system provides to its users is not influenced at all by the security measures (as long as they do not restrict access for a particular user) nor would a user detect that there are no security measures installed - security is invisible to the clients.

Technical measures: Security measures prevent potential intruders from accessing the system or the managed information through login protection, encryption or digital signatures.

Role and Relevance in Triple Space: Security is seen to be orthogonal to the other non-functional properties of a Triple Space installation. Security must be ensured on all levels, at access level to the kernel, at transfer level between users and kernels, but also between kernels and kernels, and even within the implementation of one kernel instance. In that sense, security is an ubiquitous property.

Safety: As opposed to security that is concerned with malicious influences on a system from the outside, safety considers the avoidance of failures caused by system internal components and procedures. This non-functional property is thus related to the structure and behavior of a single kernel, or at most also on the interaction of two kernels in case of data exchange and the management of shared data. Hence, safety does not have an influence on the scalability of Triple Space and was only mentioned here for reasons of completeness. According to [20] the non-functional aspects availability, reliability, security and safety are embraced by the term dependability, which in summary describes the trustworthiness of a computing system.

Technical measures: Solid and sound implementation of the internal components, and well-founded error and warning message handling at the level of individual components, but also at the level of the TS kernel.

Role and Relevance in Triple Space: Safety is very important for Triple Space, however it is mainly an internal kernel issue and hence the corresponding work is performed in related tasks of the architecture work package.

Completeness: This property is defined in relation to the ratio of correctly matched data that is returned. Ensuring completeness is particularly difficult in decentralized space installations where in the extreme case some sharing kernels of a space are not known to other sharers. Note that we differ with this definition from the view on completeness presented in [12], where it is argued that in semantic system completeness cannot be guaranteed due to the natural limits of the principle of self-representation and self-reflection.

Technical measures: In the context of Triple Space *no* distribution of data is a necessary and sufficient measure, as we consider the local query engine to return a complete result set if only locally stored data must be considered.

Role and Relevance in Triple Space: In some scenarios the completeness property is of high relevance, while in many cases - in particular when moving towards a virtually global space implementation - completeness becomes an (almost) impossible aspect to satisfy [12][17]. Completeness can only be guaranteed

in Triple Space when a closed world installation is considered, i.e. when a space is hosted by a single kernel, which provides access to the local data as a whole. In all other scenarios completeness cannot fully be ensured, and it is traded against better performance, better availability and better global scalability.

Correctness: The correctness indicator refers to the ratio of false data that was retrieved. In Triple Space these false positives reflect the non-matching statements that are returned, and hence the property becomes a matching algorithm issue.

Technical measures: A correct retrieval algorithm delivers a solution to correctness - assuming availability of a sufficiently informative set of statements that allows for correct query answering. Otherwise the matching algorithm might draw faulty conclusions and return false positives.

Role and Relevance in Triple Space: The correctness property depends on two distinct issues: i) the quality of the query resolution algorithm, and ii) the availability of a sufficiently significant set of triples. The former is not influenced by any other non-functional property, as it is fully inherited from the components that implement the matching mechanisms. The latter is influenced by the completeness property discussed above; the matching algorithm can only work properly if the available data ensures semantically correct answers, i.e. if the data set is sufficiently precise to allow the resolution of correct truth values. Completeness is no requirement, however the more complete the data set, the higher the probability of correct query answering. Hence, within the work of this deliverable, the two properties correlate.

Consistency: The consistency issue refers to the logical coherence of separately managed items, which represent the same information. The management of distributed copies of the same resource can result in consistency problems. Having the same data being managed in different kernels, that are not globally known, may result in contradictory statements being placed in the space.

Replication is, besides fragmentation, the predominant technology for addressing the challenges of distributed database systems, and is thus in Triple Space too, an interesting measure to increase the reliability and availability of information management system.

While consistency can never be fully guaranteed, as users can hardly be prevented from publishing contradicting facts, it is important that the system does not create further contradictions that eventually influence the service, and hence the functionality, of Triple Space.

Technical measures: A measure against Triple Space-caused inconsistency is the avoidance of replication-based techniques.

Role and Relevance in Triple Space: The removal or the detection of user caused inconsistencies are not a task of the Triple Space middleware. However, as mentioned above, the space installation should ensure that it does not create further inconsistencies due to varying treatment of data and spaces at different kernels. Replication-based techniques ease the implementation of highly available, reliable and fault-tolerant space deployments (i.e. dependable installations), however this increases the risk that the autonomous kernels that jointly provide such an installation end up processing and hence storing different, and

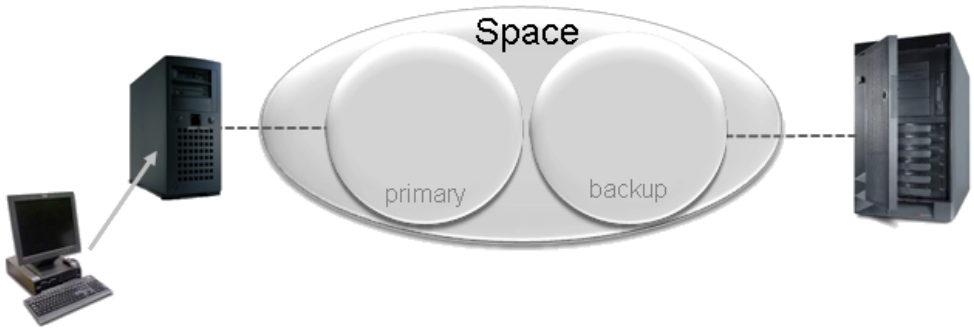

Figure 3.1: A dependable setting with minimal risk for inconsistencies.

thus potentially inconsistent knowledge. A possible intermediate solutions is the management of backup replicas that are not accessible as long as the primary copy provides the desired dependability (Figure 3.1).

Durability: Durability in transactional database systems guarantees that transactions that are successfully committed will survive permanently and will not be undone by system failure. In a more general sense, durability refers to the fact that ones data is published and stored it will not be lost, until it is intentionally removed by its owner.

Technical measures: Besides a persistent storage component, recovery is an essential means, as it ensures that no data is lost after a failure.

Role and Relevance in Triple Space: In Triple Space this non-functional property refers to the fact that the system ensures all published data to remain in the space until some other state is intentionally enforced. Durability is one of the core characteristics of Triple Space by enabling "Semantic Web services based on persistent publication of information" [11]. From a realization point of view, durability is ensured by the dependability of the kernel the data is sent to and the durability of the storage component that is bound to a kernel via storage adapter.

Performance (Response Time): Performance indicates the time a system or functional unit takes to react to a given input. Response time refers thus to the interval between a user call to the Triple Space and the time of the first response after execution, and includes system latency, as well as additional delays between a user action and the delivery of the requested functionality. Good performance is achieved by only relying on local operations, while the involvement of remote kernels obviously adds more latency. In the extreme case the other kernels must first be discovered, which, depending on the complexity of the discovery algorithms and the size of the network, can take significantly more time.

Technical measures: A first solutions is thus to limit the system size and to keep the complexity of the Triple Space low. Furthermore, load balancing – through the use of replication or distribution – is an appropriate technique to decrease the response time by reducing congestions at kernels and kernel components.

Role and Relevance in Triple Space: As for most non-functional properties, the quality of the performance aspect depends first of all on the execution time

and scalability of the TS kernel components' tasks. Therefore it is important to provide a simple implementation for the kernels that in minimal time can resolve the basic operations. On a larger scale load balancing provides an effective tool for parallelizing the client interactions on a single space. Depending on the underlying technique applied for load balancing (replication vs. distribution) such solutions can negatively influence either completeness, or consistency.

Table 3.1: Relationships between functionality and non-functional properties

	Publication	Retrieval	Removal	Notification
Availability	Y	Y	Y	Y
Reliability	Y	Y	Y	Y
Fault-tolerance	Y	Y	Y	Y
Security	Y	Y	Y	Y
Completeness	N	Y	Y	Y
Correctness	N	Y	Y	Y
Consistency	N	Y	Y	Y
Durability	Y	N	N	N
Response Time	Y	Y	Y	Y

Table 3.1 depicts the relationship between the core functionalities of Triple Space and the non-functional properties introduced. This table shows clearly that removal and notification are particularities of retrieval. They can be implemented by use of the core primitives for publication and retrieval; e.g. be publishing invalidating statements.

In the continuation of this deliverable the following properties are no longer considered: correctness, durability, security, safety and fault-tolerance. The first two are taken out of the discussion, because they are no trade-off considerations (see next section) in our setting: i) correctness is both dependent on the kernel's query engine and correlated with completeness; ii) durability is a requirement of Triple Space and will be implemented by the storage infrastructure. Security is seen to be fully orthogonal to the problem of scalability. Safety furthermore is considered to be a kernel internal issue that does not influence the scalability at Web dimension. The las property is neglected, as the issue of fault-tolerance is considered to be out of scope of the project.

3.3 Assumptions and Trade-Offs

In this section we first specify the background assumptions that the Triple Space application domain implies. This is necessary in order to proceed with a differentiated discussion of the trade-offs between the properties that were enumerated in the previous section. These trade-offs provide the basis for further scalability discussions with respect to the functional and non-functional requirements on the infrastructure.

The Triple Space platform shall provide a communication and coordination middleware that fits the following characteristics of Web: openness, distribution, scalability at Web dimensions, dynamism and decentralization.

Openness: The infrastructure is open to any contributor that is willing to host an information space. There is thus no limit to the number of spaces available, nor to the number of providers hosting those.

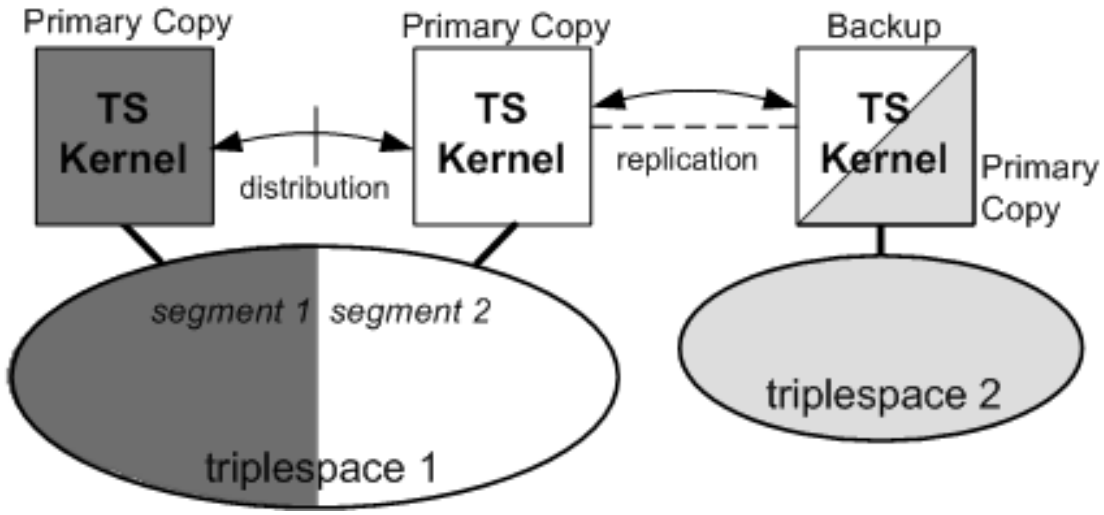

Figure 3.2: Three possible distributed realizations for Triple Space.

Dynamism: Hosts and information sources may intentionally or not go off line, spaces may be created or only temporally be put in place.

Distribution: Spaces are hosted by distributed providers running on physically distributed machines, and hence the shared information is also distributed.

Decentralization: Triple Space knows no single point of authority; responsibilities are shared amongst the hosts. A single triplespace can be shared by multiple hosts and decentralization is emphasized.

Scalability: Triple Space runs a decentralized infrastructure that scales for a undefined number of contributors (openness) and large amounts of semantic data that is distributed around the globe (distribution).

The trade-offs that influence the design and implementation choices of the Triple Space middleware at Web scale are between non-functional properties under the clear influence of the desired functionals. The basic idea being that the richer functionality the middleware provides to the client applications the less can be guaranteed about the non-functionals and about the scalability of the overall infrastructure. This process is further influenced by the technical measures applied to the realization. In principle there are three distinct possibilities that can be combined as required. The three approaches are depicted in Figure 3.2: distribution, replication and backup. The first one provides a distributed solution by managing the data of space at multiple kernels; each data item is thereby only stored on one kernel. The second one - replication - refers to the distribution principle where each kernel of a shared space manages a whole copy of the data published. Hence, every kernel stores and delivers the exact same set of data. The third approach relies either on distribution or replication, while the backup kernel is not visible, i.e. reachable by any client application (thus marked in white). This backup kernel only intervenes in cases where the primary kernel is no longer capable of delivering the expected service, e.g. after failure.

After determining the relevant non-functional properties (reliability, availability, latency, completeness and consistency) and by knowing the respective technical realizations we can now determine the trade-offs. The found trade-offs between non-

functional aspects are depicted in Table 3.2. In summary, the distribution of responsibilities and data increases the dependability of the middleware, while it decreases the quality of the data delivered. A second observation concerns the implementation approach: distribution hampers the completeness property, while replication increases the risk for added inconsistencies.

Table 3.2: Trade-offs between non-functional properties

Completeness Availability	⇔	Distributing the shared data increases the availability - the more kernels there are to access the data the higher the availability, however the lower the completeness guarantee, as not all kernels get access to all data at all times.
Completeness Latency	⇔	Distributing the shared data allows for load balancing which decreases the response time of the system, the same time - as above - distributing the data results in poorer performance with respect to completeness. This trade-off is further emphasized by the fact that the more time is invested in retrieval, the higher the probability for a complete answer set.
Consistency Availability	⇔	Replicating the data instead of distributing it allows again for multiple kernels being accessible at the same time. This increases the availability, while decreasing the consistency behavior - multiple kernels managing the same share of data (replicas/copies) might process it in slightly different ways which results in risk for inconsistencies.
Consistency Latency	⇔	As above, replication allows for load balancing at the price of inconsistency risks. The same time, as for completeness, investing more time in retrieval allows for improved synchronization and consistency checking – which basically inverses the trade-off.
Consistency Reliability	⇔	Replication delivers multiple kernels that manage the same data, thus the result sets of operations can be compared in redundant manners. Redundancy then is a core tool for increased reliability.
Reliability Latency	⇔	Comparing redundant results is costly, certainly also in respect to time. Hence, investing more time allows for more reliable results. Moreover, the maintenance of identical copies requires further efforts and resources; e.g. to keep multiple backups consistent.

Note that reliability is not in a trade-off situation with completeness, as distribution is not a measure for redundancy, and hence the reliability of Triple Space cannot be increased by distributing the data.

In addition to the relationships given in Table 3.1 of the previous section, we can now also determine the influence of query complexity and inferred data on the non-functional properties. Query resolution, depending on the complexity of the language and its features, is a particular type of reasoning - just as inference is reasoning.

As mentioned before, simple triple pattern matching can be done in distributed settings with good scalability. This is due to the nature of these project that only considers the triples are raw data units and not as semantic data or even knowledge. More complex query algorithms however depend on the interpretation of the data, and consequently rely on having a more complete view of the available data. This becomes particularly eminent when query resolution and inference is performed in decentralized

and distributed environments - like assumed for Triple Space. Therefore we also have a clear relationship between the complexity of these reasoning-based functionalities and the non-functional properties that rely on distribution of data or the retrieval functionality over non-addressable spaces.

4 Scalability Configurations

In this section the principle of Triple Space Configurations is introduced. In the first subsection we define the configuration and present the relevant attributes and their role taken from the analysis in the previous section. The second part is concerned with the application of these configurations to the scalability analysis of desired, required or available triplespace implementations.

4.1 Configurations

A Triple Space configuration provides a means to describe the functional and non-functional properties of a triplespace realization. It thus gathers the required and, on a secondary level, the desired traits of a triplespace, and puts them in priority relations. Such relations depend largely on the trade-offs presented in the previous section.

The configurations are therefore a primary instrument for the analysis of Triple Space at design time. They provide developers and space owners with structured means to express their needs, and to subsequently compare their proposed realizations with existing ones by use of the 'offline' analysis procedures describe later in the deliverable - all of course with a clear focus on scalability. A configuration does not have to describe a scalable implementation, nor a realizable one. A secondary effect of the configurations are the support that they deliver to developers in form of guidance towards scalable triplespaces.

In the continuation of this section we look at the relevant aspects of configurations and at the definition of those. A first step towards a configuration is the determination of the relevance of each of the functional and non-functional aspects of the targeted triplespace realization. In Table 4.1 the attributes that are relevant for our configuration approach are shown again, as they were determined in Section 3.

Table 4.1: Triple Space Configuration Template

Triple Space Configuration		
Functionals		Rel.
Coordination primitives		
Matching algorithm		
Data vs. Knowledge		
Transactionality		
Non-Functionals		
Availability		
Reliability		
Consistency		
Completeness		
Latency		
Further Notes		

At this stage of the project the configurations will be given in form of requirement statements and priorities that are published by use of a Triple Space configuration templates Table 6.3. These templates are applied in Section 6 to indicate the expected functionality and non-functional guarantees for the two use cases of TripCom.

The template first contains fields for the functionality of Triple Space. These are used to list the required operations, to discuss the expressivity of the matching algorithms, to require the data handling on level of triples, or on the level of knowledge, and lastly it permits to indicate the desired enhancement of the API to transactionally safe operations.

The second part concerns the non-functional aspects that were extensively discussed in the previous section. It is particularly important that it is possible to derive ordering priorities from the information given. Such priorities together with the trade-offs are needed to work out the optimal realization for a given configuration.

The "Further Notes" part provides space for further statements about the desired implementation. These are not required for the scalability-driven analysis of the triplespace realization, but might be important to understand the goal of a developer. In this part issues like security can be discussed.

Orthogonally to the three categories of attributes is the **Rel**evance column that provides a means to indicate the importance of each of the described properties. These values – 1, 2, 3, 4, and 5 with 3 being the nominal value – are of particular interest in combination with the evaluation of the trade-offs during the configuration analysis.

The filled in form is then used as input to the Triple Space scalability analysis (Section 4.2).

The TripCom project will work with at least three configurations: i) a core configuration with maximal scalability at the cost of functionality, ii) a configuration for the European patient summary scenario of WP8b, and iii) a configuration for the digital rights management use case of WP8a. The former is addressed in Section 5.2 where potential Triple Space realizations are discussed. This discussion will further show how different expectations, and thus different configurations can influence the expectable scalability of Triple Space. The latter two configurations are subsequently presented in Sections 6.2 respectively 6.3.

4.2 Configuration Analysis Method

In this section we give an outline of the configuration-based scalability analysis procedure for Triple Space. Technical details and evaluation results with respect to the different TripCom configurations will be presented in the next version of this deliverable.

The analysis procedure is based on the principle of parametric design problem solving. The idea is to assemble via configurations, and process by use of templates the parameters that define the desired or required installation that must be analyzed. The goal of the analysis is to come up with decisions on the feasibility and scalability of the Triple Space realization that matches the targeted space installation (specified by the configuration) – or at least that realizes the required triplespace as close to the configuration as possible. The principle of parametric design is to determine potential outcomes of the analysis along some common solution templates; this is possible as in any case, independent of the developer's requirements and desires, the output of the

analysis will have the same target artifacts, i.e. the same set of output parameters that provide an answer to the configuration parameters.

The input to the parametric design task are a set of requirements - specified via the configuration - and a set of constraints that are derived from the characteristics of the functional and non-functional properties and the trade-offs. The output of the method is then a set of parameters that satisfy at once the design requirements and constraints, and that maximizes the preference value by minimizing the cost-related optimization criterion. In the specific case of scalability analysis, the cost criterion is the scalability of the Triple Space implementation. Preferences differ from requirements in that they do not urge the solution to satisfy a criterion, i.e. they are solely used to judge two possible solutions as being "better" or "less good". According to [23] a parametric design method is then characterized by a mapping from such a six-dimensional space to a set of solutions designs (Table 4.2).

Table 4.2: Parametric Design Problem Specification

A parametric design application can be characterized as a mapping from a six-dimensional space $< P, Vr, C, R, Pr, cf >$ to a set of solution designs, $\{D_1$,..., $D_n\}$, where

P	= Parameters	= p1,..., pn;
Vr	= Value Ranges	= V1,..., Vn, where Vi = vi1,..., vil;
C	= Constraints	= c1,..., cm;
R	= Requirements	= r1,..., rk;
Pr	= Preferences	= pr1,..., prj;
cf	= Global Cost Function	

In order to realize the analysis method according to the principle of parametric design we need four ontologies (knowledge models) to represent the different components of the analysis tool. These models are referred to in [22] as TMDA (Task/Method/ Domain/Application), and their main goal is to reduce the complexity of the analysis problem.

Task: The task model is used to specify a knowledge level, application-independent description of the goal that shall be resolved. In other words a task ontology provides a generic means to model the task at hand; i.e. the parametric design task ontology models the different aspects that are relevant to a parametric design approach. A principle element of the parametric design task ontology is thus the concept of Parameter, and the value range, constraint and requirements models for the parameters [13]. In the context of the scalability analysis the task ontology is then populated with relevant configuration parameters.

Method: The method ontology is used as a generic means to describe the problem solving methods that are used to solve the goal/task. In particular the method ontology expresses the competence of the problem solving algorithm, defines the requirements on the domain knowledge, which is modeled by use of domain ontologies. The strength of the method ontology is to formalize the problem solving method in a highly task-independent manner; i.e. a priori independent of the parametric design task, and thus of course independent of the scalability analysis task. In [23] and [10] there are a number of problem solving methods

introduced, amongst others Extend-Model-then-Revise, Complete-Model-then-Revise (CMR), Hill Climbing or an A*-style CMR.

Domain: Domain representations deliver reusable formalizations of a particular problem domain. In our case the domain is the Triple Space. Thus the domain model provides a knowledge level description of the core entities that constitute a Triple Space installation: spaces, kernels, components, data, and of course the scalability factors and their trade-offs.

Application: The last category is also referred to as application-specific problem solving knowledge. In the case of configuration analysis the models for preferences, cost-functions, constraints and user requirements are application-specific. In other words, the application knowledge extends the other three ontologies with highly scalability analysis specific, and not generalizable knowledge. Note that some requirements and constraints are obviously implied by the domain too.

The task, respectively the method ontologies will be derived from the work that is amongst others presented in [13][22], while the domain and application knowledge will be modeled by use of the Triple Space Ontology introduced in [18] and extensions thereof (cf. for example [28]).

5 A Scalable Triple Space

In this chapter, a scalable Triple Space is introduced. It presents a Triple Space architecture in Section 5.2 where deployment requirements and choices are discussed. The Triple Space conceptual model is presented in Section 5.2. The security related issues in Triple Space are discussed in Section 5.3, whereas the persistent storage aspects are considered in Section 5.4.

5.1 Triple Space Architecture

A deployment architecture depicts the mapping of a logical architecture to a physical environment that includes interconnected computing devices. It involves sizing the deployment to determine the physical resources necessary to meet the system requirements. The selection and organization of these physical resources are guided by the requirements of the system that has to be deployed.

The assumptions and trade-offs with respect to scalability in Triple Space are defined in Section 3.3, based on which the requirements for scalable distribution of triples and their retrieval are identified and summarized in Section 5.1.2. These requirements are used as guidelines for discussing deployment choices in Section 5.1.3 and for defining the Triple Space deployment architecture in Section 5.1.4.

5.1.1 Logical Architecture

A Triple Space (TS) kernel consists of a number of components that deliver jointly the Triple Space API (TS API) [24] functionality as shown in Figure 5.1. Logically, a TS kernel resides on a single machine and can be identified with an IP address and a port number. One kernel can host multiple triplespaces. A triplespace is a logical set of semantic data that is identified by a single identifier. It is managed by a TS kernel and can be accessed through Triple Space API operations. A triplespace can be divided into virtual subspaces. The logical subset of a triplespace under a specific context is defined as a subspace, and is itself a triplespace that can have subspaces. The concept of triplespace and subspaces are defined in TripCom deliverable D6.3 [32].

The TS kernel and triplespaces are identifiable by URLs. It is possible to host multiple kernels on a single machine by allocating different ports on the same IP address to each of them. Components constituting the TS kernel are integrated by an integration middleware as presented in Figure 5.1.

Inter-kernel communication is supported by the Domain Name Service (DNS) infrastructure of the Internet. Similar to World Wide Web infrastructure, a URL address of a kernel can be resolved to the IP address of its deployment host using DNS. Kernels use the IP address plus port number of deployment hosts for their inter-kernel communication.

Triple Space clients connect to the kernels through TS API. The users of the Triple Space use the Triple Space clients to access the kernels remotely for sending and receiving data using the provided Triple Space API operations such as the *out*, *in* and *rd* operations. The *rd* operation, for example, allows reading information from a triplespace. It can be used with or without specifying the URL of the target triplespace. If the URL is not specified, the distribution manager takes care of identifying or locating

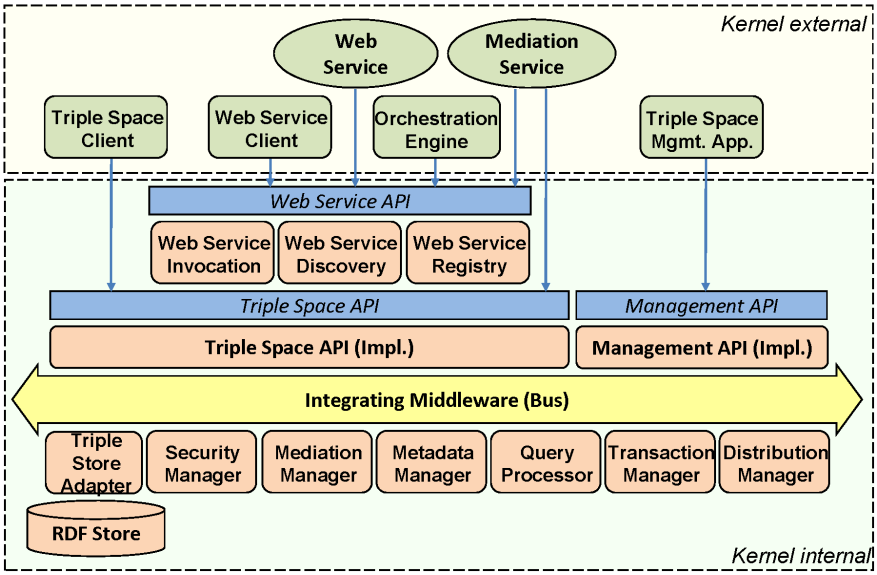

Figure 5.1: Triple Space (TS) kernel architecture

the target triplespace. For this purpose, distribution manager maintains distribution index which contains indices of the provided data and their resident triplespaces.

5.1.2 Deployment Requirements

The goal of Triple Space is to provide a scalable and distributed infrastructure with reasoning support for functional aspects such as persistently publishing and retrieving semantic data. It should support removal of data and notification of arrival of new data to clients subscribed to data matching a given pattern (cf. Section 3.1). Triple Space is to support a number of non-functional requirements such as safety and security as discussed in Section 3.2.

To support these functional and non-functional requirements, a Triple Space infrastructure may comprise a large number of geographically distributed computing resources. As stated in Section 3.3, the Triple Space infrastructure needs to provide a global abstraction layer to facilitate management and access of these resources in a coordinated manner supporting the characteristics of the World Wide Web. For this purpose, the infrastructure should ensure a number of features such as persistency of data and services, coping with system dynamism, data and service distribution, and decentralisation.

In order to support these features in a scalable manner, a distributed indexing strategy is required so that data stores, triplespaces or subspaces, and the data stored therein can be located within an acceptable time interval. It should allow distribution of data and services to support load balancing as well as failure recovery. The deployment architecture has to support flexible query expressivity as well as concurrent execution of arbitrarily large queries in order to support completeness and correctness guarantees. The data published in the triplespace can be altered or deleted; therefore, an update mechanism is required to support data consistency. As mentioned in Section 3.3, safety and security measures are equally important in Triple Space infrastructure. The deployment architecture is required to support these measures at least at the kernel level of communication. Ensuring availability and persistency of

both data and services is a crucial task that can be achieved through replication and distribution. The replication can be more useful on stable hosts. The deployment should support monitoring the online/offline behaviour of the host to preemptively decide the replication host as well as to forward queries. In addition, it has to support the mappings between Triple Space URLs to the physical identifiers, i.e. the IP and port combination.

5.1.3 Deployment Choices

The deployment of Triple Space infrastructure mainly involves interconnecting the hosts of the TS Kernels by mapping their URLs to the physical identifiers of the hosts such that they can be searched and used. This interconnection is required for using the information as Triple Spaces are hosted by distributed providers. The usual strategies of such interconnection in a distributed setting are centralised, decentralised and semi-centralised approaches.

In the centralised approach, a central index server can be used to keep track of TS kernels deployed across the network as in a client-server setting. This central index can be used to locate kernels for sharing resources (e.g. storage space) and information. This approach is simple, straight forward and provides a global view of the network, thereby supporting monitoring of participating hosts. The existence of a central server helps in supporting most of the requirements as stated in Section 5.1.2 such as enforcing security and safety measures, deciding upon replication and distribution of data and services, and coping with system dynamism. However, it is prone to a single point of failure, low performance, and poor scalability since the communication is coordinated through central index server. Implementation of a centralised index infrastructure is highly susceptible to high installation as well as maintenance costs. Google, for example, has more than 10,000 servers in its server cluster which requires significant administrative and maintenance costs [5].

In contrast to the centralised approach, the decentralised approach maintains no centralised index server for keeping track of TS Kernels deployed across the network. The network index is distributed among the participants, such that the kernels in the network maintain indices of their neighbours and cooperate to find each other. Functionally, each TS kernel in the network is equal and can perform the same set of tasks. The index is decentralised, thereby avoiding a central point of failure as well as reducing the overall network management cost. Though most of the requirements stated in Section 5.1.2 can be supported in this networking approach, it is difficult, if not impossible, to enforce security and safety measures and provide reasoning support. The scalability in this scheme is theoretically unbounded and exhibits a high level of fault-tolerance [3]. In practice, however, this scheme can only guarantee limited scalability because of its inherent characteristics such as no global view of the network, state maintenance, and inherent parallel operators (e.g., synchronisation, coordination, and network flooding).

The semi-decentralised approach is the combination of the centralised and decentralised approaches in which one or more central servers jointly maintain the network index. Individual servers maintain indices of specific parts of the network of participating TS kernels. These servers cooperate (e.g. by exchanging partial indices they maintain) to discover the required TS kernels. The index maintenance cost is dis-

tributed across these servers still achieving a global view of the network. The presence of multiple servers helps in avoiding resource bottlenecks.

In this approach, the overall performance of the systems depends on the collective performance of the servers. This allows the addition of servers to increase the system performance (and hence scalability) without hindering existing scalability configurations. Security and safety requirements can be enforced through the cooperating central servers. The semi-structured strategy is failure resilient and supports flexible query expressivity. In contrast to decentralised approaches, some reasoning support can be provided through such semi-decentralised servers.

In realising the aforementioned networking approaches, different mapping strategies (to map logical addresses to the physical addresses) can be used. In the centralised approach, DNS servers are used for mapping hosts' addresses to the human readable addresses. In the decentralised approach a commonly used mapping strategy is distributed hash tables (DHT). In this strategy, a globally known substrate of hash functions is used for mapping between the logical and physical identifiers. The same hash function is used to locate the destination of the query requests. The main advantage of using DHT-based distributed systems is their deterministic behaviour. In most approaches, search complexity is guaranteed to be logarithmic and the technique supports fair balancing of the load. DHT-based distributed systems, however, cannot deal well with "data skew" problems. Hosts storing popular data can become a bottleneck and the data is owned by the network – not the source of the data.

The aforementioned networking approaches and their support and feasibility for the Triple Space functionalities are summarised in Table 5.1. This table suggests that none of these approaches fully support the required functionalities of the Triple Space infrastructure. The semi-decentralised networking strategy, however, is the more suitable choice considering various trade-offs as discussed in Section 3.3.

Table 5.1: Deployment Choices

Requirements	**Networking Strategy**		
	Centralised	**Decentralised**	**Semi-Decentralised**
Distributed Indexing	No	Yes	Yes
System Dynamism	Low	High	Fair
Replication	Low	High	High
Backup	Low	High	High
Scalability	Low	Low	Fair
Monitoring	High	Low	Fair
Security and Safety	High	Low	Fair

In Triple Space, scalability is one of the major issues that should be supported for its Internet-scale deployment. It is observed that the network scalability is the property that depends on the underlying networking strategies. As the servers can be added, the scalability exhibited by the semi-decentralised networking strategy is best suited to support the goals of Triple Space, i.e. to support distributed coordination infrastructure for applications that credibly scale up to the size of the Internet. In the semi-decentralised approach for Triple Space deployment, kernels would serve as servers. The communication between kernels would be supported through the use of the P-Grid based indexing strategy maintained by the distribution manager. Further details of this indexing strategy is described in Section 5.2.3.

5.1.4 Deployment Architecture

The Triple Space deployment architecture is designed so that the logical operations on Triple Spaces can be translated into interactions between network addressable physical hosts. The architecture is represented in Figure 5.2. It consists of three layers of abstractions: users, triplespaces, and physical layers.

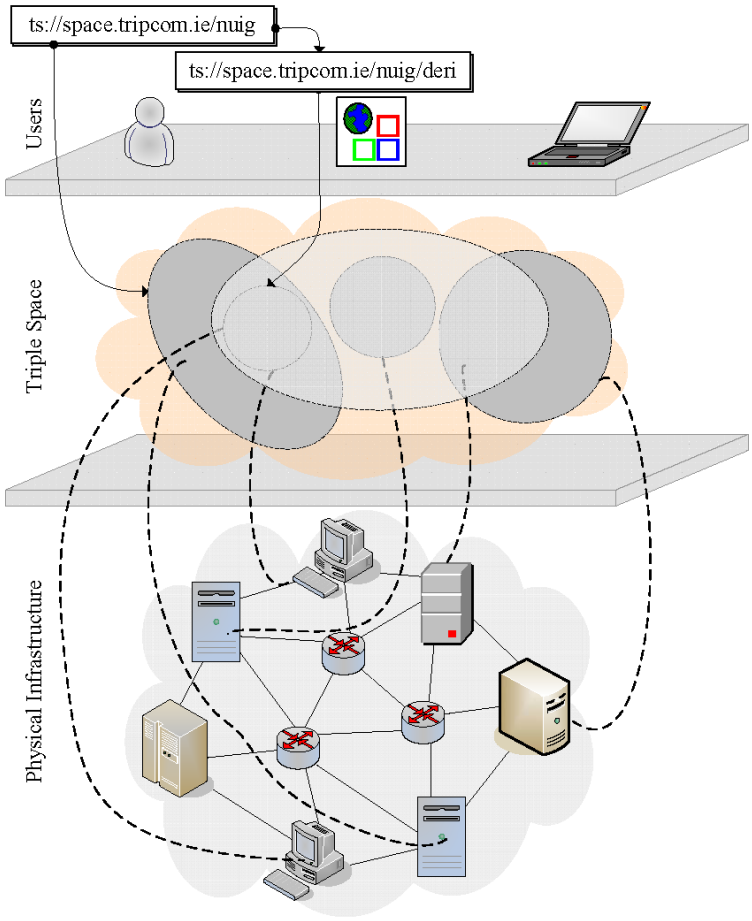

Figure 5.2: Triple Space Deployment

The user layer consists of the users and the clients of the Triple Space infrastructure. These users communicate (either locally or remotely) with the Triple Space layer through the TS kernels to access triplespaces. In order to connect to the TS kernels, users use the operations defined in Triple Space API, such as *in* and *out*.

The Triple Space layer consists of triplespaces. In this layer, the triplespaces are managed by TS kernels identifiable with URLs. These URLs are resolved to the IP addresses and ports of the physical hosts through the available DNS infrastructure of the Internet. The organisation and management of the information published in the triplespaces are guided through the use of the TS ontology as defined in [18]. This ontology provides the way to describe triplespaces and the relationship between them. A triplespace is identified by a Triple Space-wide unique identifier as in `ts://space.tripcom.org/Architecture`. The TS ontology together with the space URLs are used for building the distribution index to support inter-kernel communication. The indexing strategy is further explained in Section 5.2.3. The physical layer consists of the physical devices required for deploying TS kernels. It mainly

consists of physical entities such as servers and network endpoints. The server on which a TS kernel is deployed is identified through its IP address and port number, e.g. *172.18.254.210:8080*.

As this architecture allows addition of new hardware and software resources without hindering the existing configuration and with local indices built at each kernel, the overall performance can be kept at an acceptable level, and replication, system dynamism, openness, and reasoning at kernel- or triplespace-level could be supported. In addition, security and safety measures can be enforced through the kernels.

5.2 Triple Space Conceptual Model

Choices regarding the data and space model of Triple Space as well as the coordination model (API) were both informed by the requirement of scalability as well as impact on the realization of scalability.

5.2.1 Tuple and Space Model

Decisions regarding the chosen tuple and space model were informed by the desire to implement simplicity and flexibility while minimizing the loss of potential expressivity. We note that these choices also have a relevance to the scalability of the Triple Space.

The chosen data model for Triple Space data was RDF, which is a graph model made up of triples of the form (s,p,o). This model already provides significant simplicity, and more expressive semantic formats such as WSML and OWL languages can be serialized in terms of RDF-like triples. Hence the decision of WP2 was to specify "triples" in a Triple Space as three fielded tuples of the form (s,p,o), where s, p and o would be URIs (or in the case of the object, a simple datatype as literal), avoiding unnecessary complexities such as nested tuples or additional field constructs such as arrays (which could model RDF collections or containers). A flat (s,p,o) structure will not introduce any optimisation in the size of the triple set to represent a given RDF graph (as is typically done in backend RDF storage), but does not introduce any further redundancy beyond the standardized RDF triple representation. Note that the space data model is an abstraction also from the RDF storage used in TripCom, which does introduce optimisations. The data model restricts the possible complexity of the data being coordinated in a space, allowing other components to be optimised for a given maximum level of expressivity in the knowledge model (e.g. at present, the data model does not allow the expression of rules or axioms). Even for knowledge models more expressive than RDF such as OWL or WSML, the fact that they must be mapped to a RDF data model in Triple Space allows components, if they choose, to ignore the added expressivity of those languages and to handle the data as RDF where the scalability of the reasoning is of more importance than the added complexity of semantic expression in OWL/WSML.

The chosen space model for Triple Space considered the various space structures implemented in classical tuplespace systems, and recognized the requirement in Triple Space of at least multiple spaces (as opposed to a single centralised space) in order to support distribution. A hierarchical structure was also introduced to enable a means to restrict or extend the set of triples visible to a client, while more complex implementations such as overlapping spaces or scopes (views on data orthogonal to space structure) were avoided. Requirements collection from the TripCom scenarios

established that this multiple, hierarchical space structure was sufficient. This can support scalability in that multiple spaces allow clients to separate communication among spaces, and coordination models can be developed which attempt to minimize the communication load on a single space by splitting data coordination into sibling spaces (where there is no overlap) or subspaces (where there is overlap).

Further details to possible implementation choices and the reasons for our implementation decisions are to be found in D2.1 [32].

5.2.2 Coordination Model

Triple Space took Linda as the starting point for a coordination model. Linda is a very simple, high level coordination language which basically consists of three primitives: out, rd and in. A fourth primitive - eval - was not taken as there was no use case requirement for it, and at the same time its removal reduces the complexity of an implementation. Linda has also been extended in various implementations and there is generally a tradeoff between the expressiveness and performance of the coordination model. We have considered possible extensions and accepted in the API certain primitives as according to the use case requirements - multiple read, publish/subscribe, transactions. These issues were not taken lightly, and some difficulties of implementation in a scalable distributed system have been counteracted by a revision of the agreed guarantees made by that system, e.g. not guaranteeing completeness or consistency in the global Triple Space. Further details, culminating in a first version of the API for Triple Space, can be found in D3.1 [27].

However, a single API which provides the maximum functionality for the coordination model of a Triple Space cannot provide the maximum scalability of a Triple Space implementation because any client interacting with the space may use any of the operations of the API, even though some operations are associated with a greater cost with respect to scalability. For example, a destructive read (in) of triples is more expensive than a non-destructive read (rd) due to the need to update the (local) metadata of the kernel as well as the (global) distribution tables. Furthermore, some functionality of the original API was seen to be too expensive for the Web-scale Triple Space, such as allowing in's on the global Triple Space, and hence a refined API has been produced which can be found in D6.3 [25].

Another contribution of the API refinement was the definition of different subsets of this API to provide different levels of expressiveness in the coordination model. An API for the Web scalable Triple Space must essentially have a more restricted coordination model than the coordination model required by both use cases. Hence we have specified three APIs: one for Web scalability, one for the eHealth use case and one for the auction use case. The Web scalable API (named Core API as it is essentially a subset of the functionality of any other configuration, forming the minimal functionality that should be available from Triple Space) supports the emission of individual triples and the reading of a triple from a named space using a single triple pattern. Further extensions to this API - named Extended and Further Extended - support the additional functionalities required by the use cases at the cost of some scalability. The Extended API provides the functionality required in the eHealth use case and the Further Extended API provides the functionality required in the auction use case. The division of the API is shown in Figure 5.3.

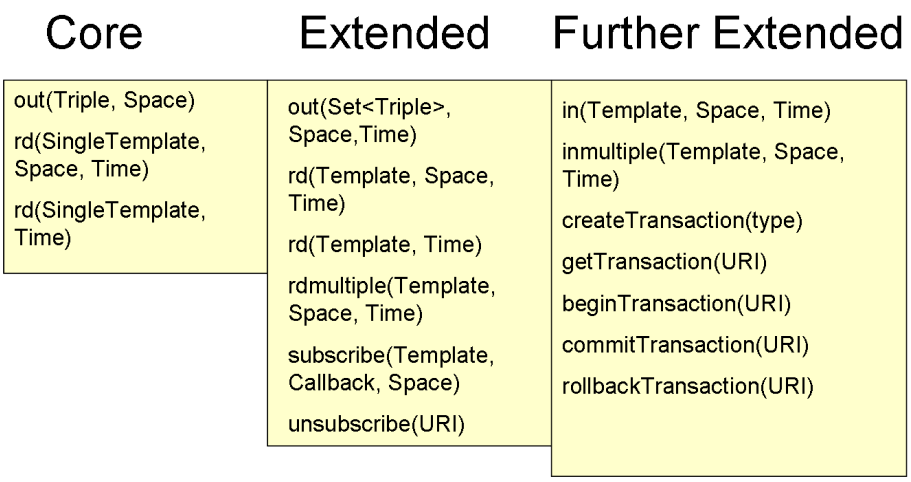

Figure 5.3: Division of the API into different levels of expressiveness

5.2.3 Distribution Model

One of the factors which have great influence on the scalability of Triple Space is the distribution strategy. From the architectural considerations in [24] we know that the Triple Space infrastructure is based on a combination system of Client-Server and Peer-to-Peer network overly. The Triple Space kernels are in a Peer-to-Peer network overly with each other and the Triple Space clients build up connections to the kernels and are in Client-Server relationship. The Triple Space clients can connect the kernels using their URLs.

The distribution of Triple Space infrastructure is based on two main strategies which are designed to handle the API operations. The first strategy is based on Domain Name Services (DNS) and the fact that each triplespace is identified by a space URL. The second one is based on a scalable P2P overlay network for providing a distributed index storage system.

To be able to describe the scalability of distribution strategy, it is needed to investigate the scalability of the distribution manager component of the Triple Space kernel and how it handles the internal processes which are caused by the Triple Space API operations. The three main operations of the Triple Space API are: out operation and read operation with and without space URL.

To handle the first two operations, the traditional name and location services are used which provide a direct mapping between the space URL and the target IP address of the deployment host. For this purpose we use the available Domain Name Services (DNS) of the Internet which is evidenced to be a large-scalable system.

To be able to handle the read operation without space URL, kernels use a distributed index storage system deployed on P-Grid [1] Peer-to-Peer overlay. The index storage system stores the indexes of user provided RDF triples together with the space URL in which these triples are available. For the processing of such a read operation without space URL, the distribution manager component of kernels has firstly to retrieve the distributed index storage system for potential spaces which are able to answer the read operation and secondly map the read operations to one or more read operations with space URL, and start processing.

The algorithm of P-Grid overlay network guarantees that queries on index storage are routed to the target peer with a logarithmic number of hops and that keys

are well balanced on the participants peers. In a network with N peers (Peers are in Triple Space infrastructure the distribution manager component of kernels), the distributed index system needs maximum log(N) hops to find the target storage. Having a logarithmic routing cost in index system is one of the factors for large-scalability of the index system. Other costs such as costs for join and leave the network are well investigated in the P-Grid project.

The other API configurations of Triple Space introduce further operations like multiple read or out operations which are implemented internally as m times single read/out operations (m is the number of single reads). In such cases, the maximum number of hops are $m \times log(N)$.

In conclusion, the distribution strategy of Triple Space is designed considering the high-scalability of the infrastructure to be able to provide services to Web-scale number of clients.

5.2.4 Querying

Query resolution on a single kernel is an issue for the Triple Space (TS) Adapter, see Section 5.4. However, query resolution in distributed kernels requires selecting which part of the global triplespace to be queried. The performance of distributed knowledge systems is strongly compromised once queries are more complex than single triple patterns; both cases of JOIN and of range queries greatly affected negatively the scalability of other systems [4].

To minimize the negative impact on scalability of non-simple queries which do not name a target space, TripCom will take two approaches: (a) for a given query (in the first implementation), determine the smallest set of spaces to query which still maximizes the chance of acquiring an answer, hence reducing network traffic; (b) for a given query (in the second implementation), decompose the query into simpler queries, and determine the simplest set of sub-queries to send to which spaces to maximize the efficiency of acquiring an answer.

The approach (a) will be implemented as an optimisation of the Distribution Manager. The approach (b) will be implemented in the Query Preprocessor.

The Query Preprocessor in combination with the Distribution Manager acts as the query optimizer for our 'query decomposition and distributed query evaluation' approach. While for this matter the task of the Distribution Manager primarily lies in fast tracking of data sources and retrieval of solutions for triple patterns the Query Preprocessor provides the modules for query rewriting, cost estimation, join and selection ordering and query decomposition and answer construction.

Before a query plan can be decomposed and distributed, the query rewriting component applies transformation rules which structurally optimize the plan for distributed evaluation. After that the query plan can be decomposed and distributed in collaboration with the Distribution Manager while estimating the evaluation costs for the partial plans by using an adaptive cost estimation technique based on an static cost model applied dynamically interleaved into the query plan execution. The static cost model itself relies on operator cost functions, i.e. a cost function for JOIN, UNION, etc., and costs indicated by certain structural properties of query plans. Furthermore we prioritize the execution of query plan parts by best-practice join and selection ordering heuristics.

The concepts described above are already thoroughly verified and well proven in the field of distributed relational databases [16, 9]. Since the vast similarity between SPARQL and relational database query languages on the algebraic level [7] we imply that benefits gained by those measures also apply to SPARQL: it is to emphasize that the adaptivity loop which is applied periodically during a query plan evaluation moderates the exponential growth of the cost estimation error in comparison to an upfront estimation for the complete query plan. With this dynamic cost-based distribution of (partial) query execution plans to available resources (CPU, I/O) of kernels we can generally achieve scalability for query processing of arbitrary complex queries. Further details on the Query Preprocessor can be found in D3.3 [26].

5.2.5 Discussion on Scalability, Distribution and Querying

The previous sections have shown that the scalability issue is mostly apparent in the context of the retrieval functionality of Triple Space - see the discussion on rd over a distributed Triple Space in 5.2.3 and on complex query handling in 5.2.4. On the one hand scalability is more tangible at retrieval time also for users, and on the other the influences of semantic computing (e.g. inference) are primarily applied at retrieval time. Still, it is impossible to separate the realization of the two core primitives. The behavior at publication time directly influences the necessary effort and the quality of service when reading from a triplespace. Nonetheless, for the remainder of this deliverable we concentrate on the impact that the various retrieval operations have on the expected scalability of the Triple Space - also motivated by the fact that this can trigger more concrete discussions about TripCom's influence on novel scalable and dynamic reasoning techniques.

The retrieval primitives defined in the course of the TripCom project result in three interaction models with clearly different prospects in what concerns scalability.

The most promising approach - promising with respect to scalability, completeness and consistency - is based on access to a triplespace by `rd` with a specified target space URL. Retrieval is then empowered by locating the desired space, given by the URL. DNS-like discovery mechanisms, analogous to the Web, provide a feasible solution. As DNS has proven to scale, this approach to retrieval should also scale.

Triple Space access by `rd` without specified target space URL will certainly create more concrete challenges. First, the relevant sources must be discovered within the open collection of spaces, and second, the data from potentially multiple sources must be combined. This becomes particularly challenging when choosing a knowledge-driven approach to Triple Space Computing - as the algorithms must reason over multiple distributed sets of triples. We distinguish two different approaches to read without indicated target space:

1. The first approach - also referred to as search - aims at the retrieval of indicative information and pointers to additional relevant data and sources of information. In that sense this approach is expected to scale too, as there is no requirement on completeness or consistency of data. This is hence a best effort retrieval solution that does not require any knowledge about relevant sources (the space URL) from the clients, while conforming well to the recent trend in knowledge retrieval expressed in [12]. It can even be assumed that the middleware only deals with explicit data, and distributed query resolution and reasoning are not a must. Consequently the scalability is only in a trade-off relationship with the

performance property, which in turn depends on the discovery algorithms that manage the internal links to relevant sources and the forwarding of requests.

2. In contrast to the previous setting it might also be required to deal with knowledge and return inferred data too. This type of retrieval relies thus on a certain degree of completeness. The predominant problem is thus how to achieve completeness, or when to consider the collected data as quasi-complete. A very important issue is the realization of the discovery procedures: where to end the routing of requests? Should only information be retrieved that is locally stored, is there a fixed number of hops for the search path defined? Even so potentially realizable and effective, these solutions do by no means ensure completeness.

   An even more important trade-off relates to the aforementioned issue with query resolution and reasoning. A read call without target space collects arbitrary matching data that has to be interpreted as a whole in order to resolve the relevant truth values. In the general case this calls upon distributed reasoning over all relevant kernels, or large scale (local) reasoning after transferring the data to the local kernel, where the request can be handled in its entirety. For the time being, it seems thus impossible to accomplish true global knowledge retrieval methods without paying the prize of incompleteness. By refraining from the simple scalability measures that Triple Space provides by naturally grouping related data in triplespaces, this retrieval method enforces a trade-off between response time and completeness.

In Triple Space, we have implemented querying in distributed spaces without the guarantee of completeness, rather we use Distributed Hash Tables not to answer the query (as in other distributed knowledge systems) but to answer the question, which space can I query to get an answer? Hence we can combine the scalable properties of distributed hashing on single triple patterns with query answering for more complex queries with inference, in that we select spaces based on the most "selective" triple patterns in a query and assume the appropriate schema information is found at the target kernel. Likewise, we relax the requirement of consistency, not assuming consistency between spaces on different kernels and applying principles of inconsistent reasoning to spaces within kernels to approximate answers where complete schema information is not available. Hence, we solve inference problems with inconsistent or incomplete knowledge with a minimal cost to scalability by only handling the problem locally. In the next phase of TripCom, we also introduce ideas of space distribution over multiple kernels to allow data to self-organize closer to related data, as well as a process of query cost measurement and decomposition to allow complex queries to be broken down into simpler parts and answered at different kernels. Hence we will take measures to handle the inherent incompleteness and inconsistency of knowledge in the Triple Space by forming clusters of kernels/spaces which maximize the completeness of their contained knowledge as well as splitting the retrieval operations across kernels which are individually maintaining consistency in their local knowledge. These approaches form a means to further improve the efficiency and accuracy of semantic queries without incurring high costs in scalability.

5.3 Triple Space Security

Conceptually, security acts as a filter that lets through only allowed operations, according to the security policy. Apart from a kernel that hosts only spaces accessible to any client for any operation, security checks are applied to every request that a kernel receives. Therefore, it is crucial that security checking does not constitute a bottleneck for the system.

Security of the communication channel between a client and a kernel is achieved using the well-known TLS standard protocol, which requires asymmetric cryptographic operations[1] only during the handshake (TCP connection establishment[2]) rather than for each message. It is expected that a client may issue many subsequent requests to a kernel in order to perform a task; therefore keeping some state allows to optimize performance because it avoids repeating heavyweight operations at each request. According to this principle, the design of the security functionalities includes the *cookie* concept, which is used by the Security Manager kernel component to refer to state information that is kept between different requests from the same client.

Access control policies are based on *roles*, which can be defined by applications according to their needs. On the other hand, the client can provide a set of more general-purpose *attributes*, contained in SAML assertions issued by attribute providers. The client attributes are then mapped to access control roles using attribute mapping rules, which are part of the triplespace policy. This design has the following advantages:

- it allows the policy designer to focus on application needs, formalizing application-dependent roles and using only these roles in the policy, rather than general-purpose client attributes or system-defined roles. In this way the policy can be more compact (thus quicker to evaluate) and easier to design and understand;
- it does not require to have all clients / users registered into the kernel.

The policy model includes a choice of different policy combining algorithms, which regulate the order in which policies are evaluated in the hierarchy of subspaces, and which policy prevails. This mechanism, if used properly by policy designers, avoids the need to evaluate all policies in a hierarchy of subspaces (which could be costly if the hierarchy includes many subspaces and thus many policies).

If a client issues multiple requests to the same kernel in a short period of time, there is no need to repeat the signature verification on client's certificate and assertions. This is a computationally heavy operation, so it is performed for the first request and then the results are stored in a temporary *security context* inside the kernel. The security context identifier is returned back to the client (the cookie); for the following requests the client can just send the cookie, so that authentication and assertion validation phases can be skipped. Moreover, if the target space of the request is the same, also the results of the trust and attribute mapping process will be the same. These results are thus also stored in the security context, and the trust and attribute mapping process can be skipped or reduced for subsequent requests[3].

[1] From a computational point of view, asymmetric cryptography is significantly heavier than symmetric cryptography.

[2] TLS also includes a session resumption mechanism.

[3] The results depend on the target space and on parent spaces, so partial or even complete results may be available even if the target space is not the same. See deliverable D5.2 for details.

From an implementation point of view, many optimizations to improve performances and scalability can be introduced. The security context is designed so that it can be stored in the system bus, rather than inside the Security Manager kernel component, in order to allow the possibility to have multiple equivalent instances of the Security Manager in a single kernel. Furthermore, the Security Manager is designed so that the logical subdivision between its three main functionalities (i.e., Authentication, Trust and Attribute Mapping, Access Control) can also be mapped to a physical division into three separate subcomponents, if needed. These sub-components can moreover be replicated and even "specialized": for example there could be different Authentication subcomponents running in parallel, and in case they can also be dedicated to different authentication mechanisms[4]. All these options can be exploited if one kernel needs to be distributed over different machines.

5.4 Triple Space Persistent Storage

The persistent storage is responsible to ensure the durability of the Triple Space information model and all internal component data in the boundaries of a single TS kernel. Strictly speaking, the functionality provided by the persistent storage partially overlaps with the features of the Triple Space if all problems related to coordinated access to distributed semantic data are ignored. Thus, the proposed scalability definition in Chapter 2 are valid if they are revised to the context of a single TS kernel instance.

A typical semantic or triple store engine is similar to a database management systems (DBMS) to allow for persistency querying and management of structured data. The major difference between the two systems is the way the data is interpreted, so the triple stores perform all its operations regarding the semantics of ontologies and metadata schemata. Hence, a principally new type of functionality emerges - the support of efficient reasoning. The task of reasoning nowadays is often considered together with the storage of semantic data, because the very tight integration between the storage and inference engines imposed by the requirements for efficiency. From that point, we will consider the tasks for persistent storage of semantic data and reasoning as invariably coupled.

To trace the system scalability we have to investigate how the performance of the two core functions - data persistence or modifications and retrieval relate to storage scale. Data modification is a process to load and index new information or delete previously asserted facts. Data retrieval is the process to read persisted information and derive implicit knowledge. The scale of semantic stores is expressed by the number of asserted triples compared to the change in the complexity to perform the two operations. Hence, to complete out analysis we should consider also two criteria which greatly affect the complexity to implement semantic store:

Semantics expressivity: The supported ontology language by the semantic storage is critical to the rational between explicit vs. implicit asserted facts. Even the simplest fragment of OWL (OWL-Lite) is not tractable which results in non linear complexity and imposes serious scalability limitations. The supported semantics affects the data modification and retrieval time by:

[4]The current implementation relies on client certificates, but nothing in the model prevents the introduction of other authentication mechanisms.

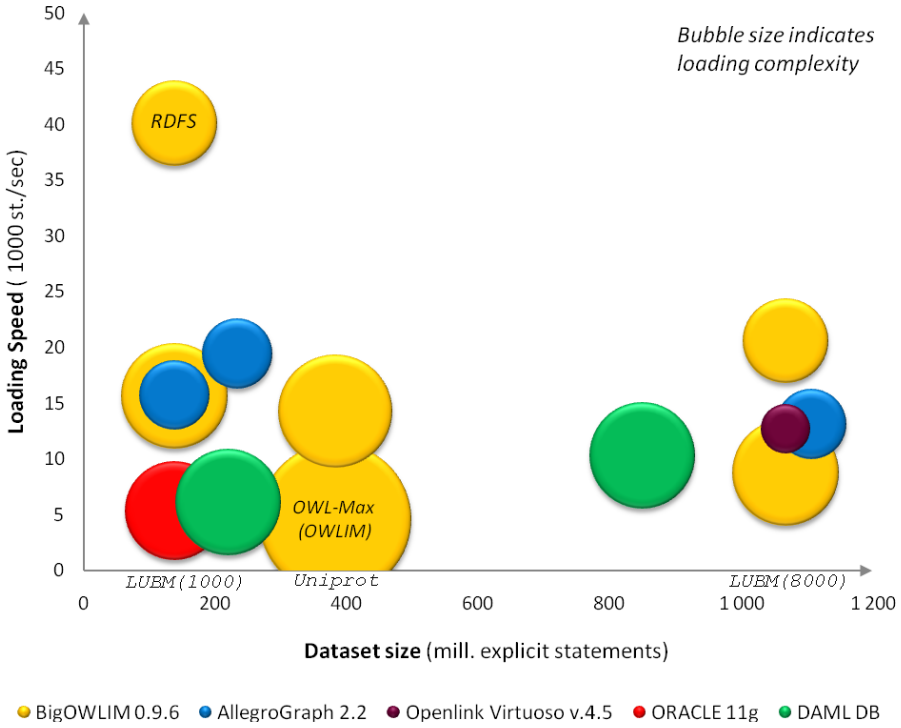

Figure 5.4: Public results to present scalable storage and reasoning (bubble size indicates loading complexity) [15] .

Data modifications are affected by the degree of materialization and pre-calculation of the inference closure during write time i.e. to what extend forward-chaining algorithm is used. For instance, system may load millions of statement using simple semantics and in the same time not being able to compute the inference close of several hundred statements.

Data retrieval time depends on the degree of deduction, whether the inference is performed query time. In backward-chaining evaluation systems.

Complexity of data model: Many different extension of the RDF data model are introduced where the most notably with respect to the available triple store implementations are named graphs [6] and triplesets [21]. Richer data models are more resource consuming. In the next paragraphs a deeper analyze is presented, how this affects the overall scalability. Other important criteria are transactional support and query language expressivity typical for every storage system. Operations like joining, negative related clause or full-text search obviously requires more computational complexity and also impact the system scalability.

Kiryakov in [15] summarizes the state-of-the-art storage implementations and correlates the scalability of the systems according to the supported semantics, data loading and retrieval time. Figure 5.4 demonstrates data loading vs. supported semantics, where the complexity of data model is considered only as triple compatible model to support SPARQL.

The Triple Space data model is bound to RDF and syndicates information derived from multiple persistent stores. The highly distribution nature of the model inevitable leads to number of problems related to restricted information usage, access controls,

logic scoping, data ownership and etc. All the explicit client application data must be associated with a number of implicit meta-data generated by the Metadata Manager about the client's user, space, operation context, timestamp and so forth, and then it is mapped to persistent store model and saved. Theoretically, RDF data model expressivity is enough to represent the client's data and its meta-data, however such approach would not be feasible because the higher complexity during all data modification operations to preserve its consistency. Lastly, if the TripCom architecture treats the client's application data and the system data equally malicious users may try to alter it. TripCom persistent storage model has increased complexity [21], which will significantly reduce the amount of triples needed to represent the Triple Space information.

The scalability of semantic storage system is dependent on the concrete requirements of the usage scenarios and may vary in order of magnitudes. TripCom persistent store architecture is designed to accommodate these requirements and to support different configuration levels with respect to data model representations and semantic expressivity, which maximize the persistent store scalability. A more sophisticated study on the equivalence of the implemented model scale compared to RDF data model scale will be placed in the TripCom deliverable D1.4 "Storage Performance Evaluation".

6 Scalable Realization of Use Cases

In this chapter, we elaborate on how and to which extent, at which level can Triple Space ensure that the use cases can be realized in a scalable way.

6.1 (Semantic) Web Services

TripCom and the TS API provide relatively low level functionality to the users, e.g. description, discovery and invocation of remote functions has to be done manually. This is were the Web service integration part comes into play: it provides complementary functionality to the Triple Space API, providing high-level access to Web services. Details about the architecture are presented in [29] and summarized below:

Web service registry: Currently, Web service descriptions (e.g. WSDL files) are made available in a Web service registry for easier management and to facilitate reuse of functionality. However, these registries are often company or even product internal; global Web service registries do not exist. TripCom provides a global-scale infrastructure for communication of semantic data, thus, by implementing a Web service registry using TripCom technology as the underlying infrastructure, a truly global Web service registry is created. The implementation is based on the UDDI [33] datamodel and interface, providing transformations from the UDDI datamodel to RDF and back in order to store Web service descriptions using TripCom.

Web service binding: As the second point of Web service integration, TripCom provides an implementation of a *Web service binding* that allows using TripCom as a transport mechanism for Web service communication [30]. TripCom as a Web service transport provides unique Quality of Services such as the native support for extended Message Exchange Patterns (MEP) to clients using this binding. Furthermore, clients using RDF as data format do not have to transform to another (semantics unaware) XML representation and then back to RDF at the server side as they would need to do using traditional Web services (i.e. lowering and lifting). Using the TripCom binding, Web service requesters and providers that communicate semantically enriched data do not have to leave the semantic layer, RDF is communicated directly using a semantics aware transport. In contrast to the TS API no additional efforts are required to invoke (remote) services, Web service invocation is directly supported through the binding.

Integration with the Web Service Execution Environment: The Web Service Execution Environment (WSMX) [14] is an execution environment for Semantic Web Services. The interaction model supported by Triple Space Computing envisions a new communication paradigm that Semantic Web services can utilize for achieving scalability in Web-scale communication and storage. Triple Space Computing provides a uniform coordination model which can be used by WSMX to perform asynchronous communication (regardless of time, space and reference). It helps WSMX to become a Web scale middleware for Semantic Web Services. It will bring several benefits for WSMX with respect to its communication and storage management [31]. From communication perspective, the integration with Triple Space will help WSMX in improving the reliability

and robustness in its internal component management, external user interaction and inter-WSMX communication, rather than always sending direct synchronous messages and wait for the response. From storage perspective, it will help WSMX by providing it a globally accessible persistent storage for storing its semantic descriptions of web services, goals, ontologies and mapping rules, rather than for every WSMX instance to maintain separate repositories.

The Table 6.1 provides formal description of the functional and non-functional properties of Triple Space required by Semantic Web Services.

Table 6.1: Web Service Triple Space Configuration

Triple Space Configuration		
Functionals		Rel.
Coordination primitives	Coordination primitives are important for WSMX, in order to (1) access the Triple Space, i.e. to retrieve, publish, manipulate and delete triples, (2) subscribe to sub-spaces to receive be notified and (3) manage the triple space	4
Matching algorithm	Accuracy of matching algorithms is important to retrieve exactly what is required by WSMX	4
Data vs. Knowledge	No high requirements for inference, as WSMX use reasoners for specific DL, rule and other reasoning in service discovery, selection	2
Transactionality	Required. Data to be stored by WSMX to TripCom has to ensure that it has been successfully stored; otherwise it can loose the information about available ontologies and semantic descriptions.	4
Non-Functionals		
Availability	After enabling WSMX storage and communication using TripCom, WSMX gets certain level of dependency on TripCom and requires it to be highly available in order to carry out its communication with users, end-point services, and retrieving ontologies and semantic descriptions from Triple Space.	4
Reliability	Retrieved information from the underline TripCom middleware have to be reliable enough to for WSMX to be able to produce	4
Consistency	The retrieved information should be consistent in order to enable WSMX create correct (and not invalid) results for service discovery ... execution.	3
Completeness	WSMX also required completeness of the data retrieved in order to discover complete set of services, and not to miss some information that might be of use.	3

Latency	While achieving a-synchronicity of communication and storage management in WSMX, latency from underline Triple Space can be tolerated to some extent	2
Further Notes		
Security	Ontologies and semantic description of services are normally used for public exposure for the users to be able to search for services world-wide. However, security issues concerns authentication and authorization for the users invoking Web Services with confidential data (i.e. credit card information).	2

6.2 eHealth

The eHealth scenario implemented in TripCom aims at demonstrating the TS capabilities to work as semantic middleware for supporting a Patient Summary infrastructure at European Level. Such a scenario, named the European Patient Summary scenario, demands strong requirements in terms of scalability since the supporting infrastructure has to be able to manage the patient summaries of each of the over 500 million European citizens.

The EPS space is virtually one big space within the global triplespace infrastructure. In reality, it is subdivided into many subspaces following the natural structure of health authorities in Europe. There are several nations with national authorities which in turn are subdivided into regional, county and city authorities. Although the number of levels may vary from country to country the overall structure remains the same. In a similar manner the EPS space is represented by a tree of subspaces with the root representing all of Europe, and the leaves delivering the individual citizens' patient summaries as shown in Figure 6.1.

The security policies of each of the subspaces must be configured accordingly to the privacy regulation for the treatment of citizens' data. Moreover, each citizen has to be able to restrict the access to certain specific content of his/her summary. For this reason, every summary is further divided into two subspaces: one for the public parts of the patient summary that is protected according to public regulations, while the other one is protected by the citizen's specific access policies.

Each local authority is responsible for managing the data related to the citizens that live within its geographical boundaries. We assume that each authority provides at least one kernel to the EPS infrastructure for supporting the summaries of the managed citizens. Within Europe there are large health authorities that manage more than one million citizens and numerous smaller authorities that manage a few hundred thousands citizens spread over their geographical territory. Considering that the bigger authorities are probably subdivided in several sub-sites each having its own kernel, we assume that the numbers of kernels available for the whole EPS is about 5,000.

Taking the previous numbers, the following table (Table 6.2) reports the dimensions of the EPS scenario both at the level of the full infrastructure and at the level of a single Triple Space kernel.

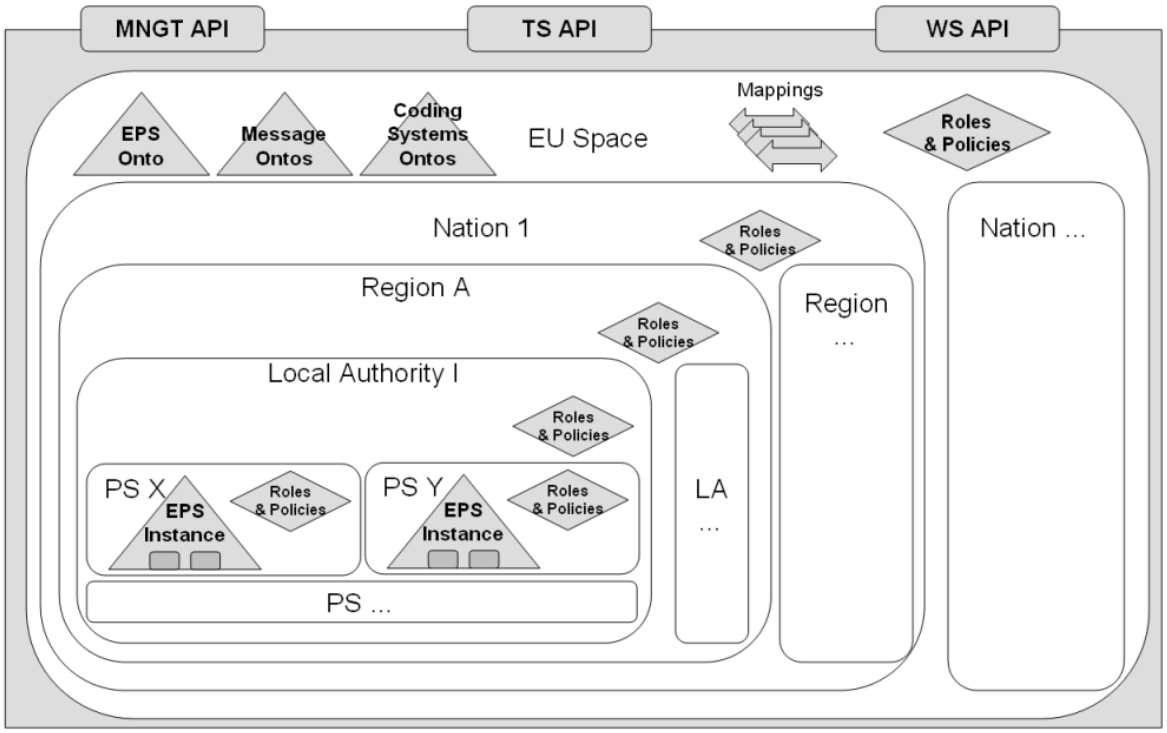

Figure 6.1: The hierarchy of subspaces in the Triplespace

Table 6.2: The dimensions of the EPS

Full Triple Space Infrastructure	
500,000,000	patient summaries
1,000,000,000	subspaces (∼2 per summary)
1,000	avg. number of triples per summary
500,000,000,000	triples in the overall system
Per Triple Space Kernel	
100,000	patient summaries
200,000	subspaces
100,000,000	triples managed on each kernel

The Table 6.3 reports the description of the functional and non-functional properties of the Triple Space required by the EPS scenario accordingly to the scalability configurations described in Section 4.1.

Regarding the functional aspects, the EPS scenario requires the *coordination primitives* to publish data into the Triple Space and retrieve it with time, location and reference autonomy. Moreover, it is useful to permit healthcare actors to subscribe on particular subspaces and operations. Although the EPS scenario deals with healthcare data and no deletions should be permitted in general terms, it can be useful to permit such operation for administrative and maintenance reasons (such change of registry data or errors).

The healthcare providers act as actors of the Triple Space and have to be able to extract the whole summary of a citizen or a subset of it by submitting queries that need to be answered by the *matching algorithm*. The previous queries retrieve data from a specific set of subspaces. Other queries, such as the statistical queries, should

be able to retrieve data from multiple subspaces.

For what concerns *data vs. knowledge*, the EPS scenario uses inference for two reasons: the semantic mediation between the internal EPS ontology and the eHealth standards and the gathering of data for statistical purposes. Even though useful, those functionalities are secondary to the coordination primitives and the matching algorithm.

The scenario requires publishing a set of triples in a one shot operation. For this reason, *transactionality* is not a mandatory feature.

The primary scope of EPS infrastructure is supporting the continuity of care of the citizen when he/she is abroad or moves to another authority in Europe. For this reason, high *availability* is expected, even if a disconnection for a limited period of time can be acceptable in non-urgent situations.

Due to the management of health information, the EPS scenario requires that the data managed in the Triple Space remains correct, *reliable* and *consistent* as published by the healthcare providers and that the data is retrieved *completely* when a specific citizen's summary, which is stored in a limited set of subspaces, is requested.

When not dealing with emergency situations, the *latency* acceptable for the EPS scenario is measurable in terms of a few minutes for discovering the subspace of the citizen and retrieving his/her summary, accordingly to security regulations.

Finally, the EPS infrastructure has to guarantee that only authorized users have access to patient data so that *security* in accessing data plays a crucial role in the scenario.

6.3 EAI

The EAI scenario described in [8] aims at implementing a Digital Asset Management (DAM) marketplace, where several Content Providers offer content by persistently publishing content catalogues in the Triple Space. Service Providers want to offer content services to their customers, and purchase the needed content by calling open auctions where Content Providers compete to offer the requested content, improving other's offers in terms of quality and price. In order to provide requirements for TripCom scalability, we will assume the aforementioned scenario will be deployed in a commercial infrastructure.

The space hierarchy shown in Figure 6.2 contains the space structure to store the more relevant information of the marketplace application. The content catalogue is structured in a sub-catalogue hierarchy, where every Content Provider might define its own sub-catalogues in order to control the content offered by itself (avoiding thus the malicious information deletion by competitors). The same philosophy applies to service catalogue and auction management, where each Service Provider controls its own service offers and auctions. Additionally, contracts are stored so that each pair of signers control a shared container to ensure their contracts are only modified by themselves.

The complexity of this space hierarchy heavily depends on the number of actors (Content and Service Providers) involved, which is the same behaviour than many commercial systems, where the amount of clients and applications define the complexity of the system. Contrary to the EPS use case, where the amount of information can be predicted, or at least bounded, in a commercial deployment of a DAM marketplace, we can only expect a growing size of both information and complexity. The relevant

Table 6.3: Triple Space Configuration for the EPS scenario

Triple Space Configuration		
Functionals		Rel.
Coordination primitives	Out set of triples, rd with template and rd-multiple are fundamental. Subscribe and unsubscribe are useful. In-multiple may be useful.	5 4 3
Matching algorithm	Matching for answering SPARQL queries or graph patterns within a limited set of subspaces is fundamental. Statistical queries are based on SPARQL queries and should be able to retrieve data from multiple subspaces.	5 4
Data vs. Knowledge	Inference features may be useful.	3
Transactionality	The publishing of a set of triples is performed in one shot and there is a weak need for transactionality between multiple actors.	2
Non-Functionals		
Availability	Needed to support the continuity of care. Can be relaxed in non-emergency situations.	4
Reliability	The system must provide correct results in any situation when it's available.	5
Consistency	The triplespace must not alter the consistency of the data published.	5
Completeness	Fundamental for a single summary. May be useful for statistical reasons on multiple summaries.	5 3
Latency	Response time of some few minutes in non-emergency situations.	3
Further Notes		
Security	Need to guarantee that only authorized users have access to the data of the citizen.	5

metric for this use case is ensuring that as complexity grows, the performance of the system is maintained by adding the needed infrastructure. This means that the priority for Triple Space scalability will be focused on maintaining the functionality with acceptable response time in this use case, even if the resources needed are not linear. The reason for this decision is that while costs do not grow too much, a company will be interested in providing a service to as many actors as possible, maintaining the service's quality.

The EAI use case will make use of basic Triple Space functionalities (publication and retrieval of information), extended functionality (multi reading and subscription mechanisms), and eventually further extended functionality (transactions), as described in Section 5.2.2. Transactions will be used in order to assemble services and take care of contract signing. Additionally, the use case defines a set of non-functional requirements which are closely related to the scalability of the Triple Space infrastruc-

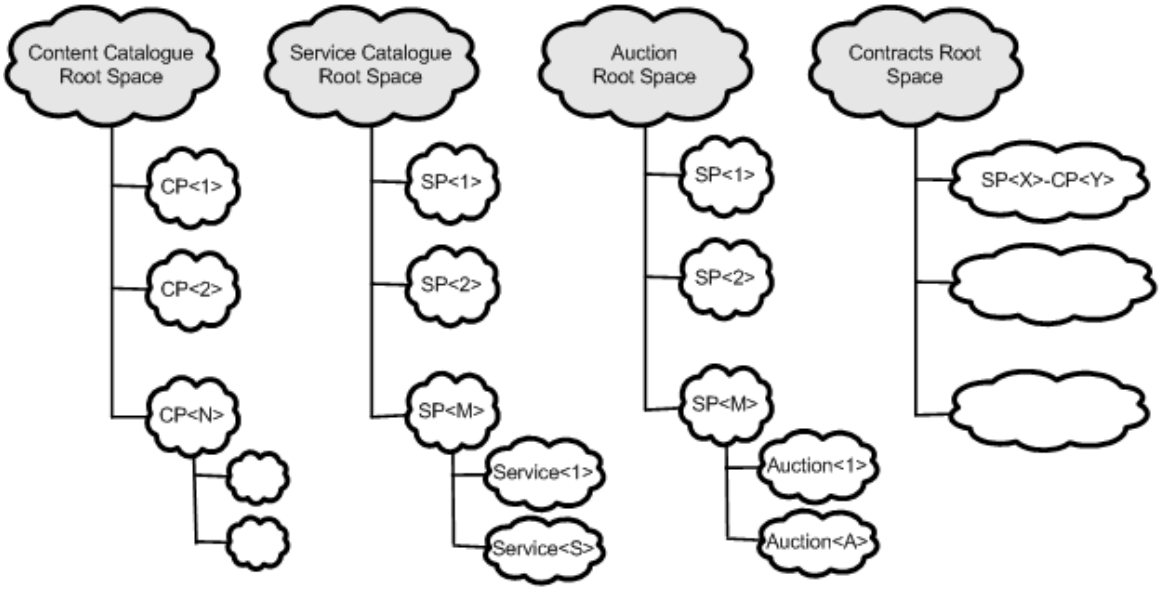

Figure 6.2: The Space Hierarchy of EAI Use Case

ture A summary of both functional and non-functional properties and their impact on EAI use case, can be seen in 6.4.

Table 6.4: Triple Space Configuration for the EAI scenario

Triple Space Configuration		
Functionals		Rel.
Coordination primitives	Simple and multi reading primitives are needed in order to store and retrieve information. Subscription, unsubscription and notification means are also needed in order to handle the communication between service and content providers and moderate the auctions between them.	5
Matching algorithm	Template matching mechanisms limited by timeout parameters are needed, and should be precise in order to retrieve the needed information. SPARQL queries are also needed for more complex information retrieval operations.	5
	Statistical queries based on SPARQL queries could be useful in order to provide some statistical information to the marketplace manager.	3
Data vs. Knowledge	Although knowledge is not needed in order to implement the basic functionality needed by a marketplace, inference features can support several added-value functionalities that be very important form the exploitation point of view. For example, reasoning capabilities can enable the implementation of effective cross-selling strategies and advertisement.	4

Transactionality	Transactionality can be needed in order to ensure that all the steps of a negotiation are completed and the contract is fully defined between two actors, in order to avoid inconsistent views of a contract. It would be more important in a exploitation deployment of the use case, so it won't be given a top priority for the use case development.	3
Non-Functionals		
Availability	Data availability is not a critical issue for this use case. Although data is expected to be returned in a reliable way, short disconnections and data unavailability can be tolerated by the system.	3
Reliability	The system must provide correct results in any situation when it is available.	5
Consistency	The previous requirement also involves that information retrieved should be correct. Although the managed information is not critical, it is highly desirable that only Content Providers which match a Service Provider's expectations are returned in order to automatically start an auction process inviting suitable Content Providers.	4
Completeness	The design of the use case makes necessary that the Triple Space infrastructure returns all matching information from a concrete catalogue space. For example, a customer is supposed to be able to retrieve all the content services that match its requirements.	5
Latency	Performance is a very important factor for this use case. It is expected that response time allows the agile query of different catalogues, and that the number of actors can easily scale, maintaining the response time and functionality.	4
Further Notes		
Security	The DAM marketplace has to ensure that only sub-catalogue creators can modify their content or service offers, avoiding the malicious deletion of relevant information.	4
	The space hierarchy defined for this use case makes necessary to provide the DAM marketplace application suitable means to create and delete sub-spaces, allowing each actor to define as many sub-catalogues as it wishes if it has enough permissions to do so.	5

7 Evaluation of the Prototype

7.1 Evaluation Procedure

In this section, a structured approach to evaluating TripCom with regard to scalability is presented. As already stated in Chapter 2, "scalability for Triple Space can be defined as a property which allows a triplespace system to maintain acceptable performance by expanding in a graceful way by adding components to handle increasing demand ...". In particular, a system that can "balance an increase in demand with a proportional increase in hardware", and "whose performance improves after adding hardware, proportionally to the capacity added" is considered (linearly) scalable. The latter of these two conditions does not take into account an increase in demand, but observes how performance behaves with varying ressources, whereas the first one describes the simultanous increase of hardware resources and demand, and requires the performance not to be significantly affected.

Consider an example where the doubling of the number of clients (assuming a constant demand per client) can be perfectly compensated (i.e., response time does not increase) by doubling the number of machines over which a space is distributed (so long as bandwidth constraints are not reached). Such a behaviour can be called "ideally" scalable in clients vs. machines (under a given set of constraints).

In general, we can describe quantitative aspects of scalability on the basis of three dimensions, namely

- the computational resources available (expressed, e.g., by the number of machines),
- the demand (workload, or simple "load") on the system (expressed, e.g., by the number of clients), and
- some performance measure (e.g., the response time).

Notice that usually performance (in the following denoted by P) is *decreasing* with load (L), when resources (R) are held constant, e.g., response times increase with the size of the data transferred, and is *increasing* with R, when L is held constant, e.g., response times decrease with more physical machines.

A particular aspect of scalability is captured by the answer to the question, how P is affected when more resources (larger R) have to compensate for more load (larger L). A constant remaining value of P when simultanously increasing L and R by the same degree would express "ideal" scalability. This restricted aspect of scalability of a load against a resource can also be quantified by means of a ratio of the performance under (small) reference load and resources and the performance under proportionally increased load and resources—given maximal utilization of resources in both cases.

Similarly, the values of other parameters, characterizing different aspects of resources (e.g., the number of physical machines running the kernel, memory size, the number of cores in a multi core processor, clock speed, cache size; interconnection bandwidth), load (e.g., demand for memory, bandwith, and operations as mentioned in Chapter 2, but also the frequency of requests issued by the clients, and the volume of data passed from/to the clients), and performance (e.g. throughput), can be related to each other and thus provide insight into the scalability behaviour of TripCom.

By defining a set of load parameters, a set of resources parameters, and a set of performance measures, one lays the ground of a model of the system's behaviour with respect to performance. Resource parameters reflect both the "scaling up" (e.g., via higher clock speed) and "scaling out" (e.g., via running the system on four machines instead of one) approach. The relationship among the demand on the system, certain system resources, and a performance measure can be characterized—under certain constraints—by quantitatively capturing, i.e., measuring, the relevant values in the course of experiments. Correspondingly, the scalability evaluation of TripCom comprises the following steps:

1. specify performance model parameters (relevant parameters to describe load, resources, performance measures),
2. review the performance model,
3. collect performance data (relationship among performance, load, resources) via experiments:
   (a) design experiment (specify test cases),
   (b) implement performance measurement program,
   (c) generate test data,
   (d) run performance measurement,
4. analyze results.

Simulations of the performance behaviour on the basis of a model of the system are not regarded adequate; in order to yield realistic results, the model needs to be very fine-grained and thus enormously complex. Besides, such a model is not available.

7.2 Performance Model Parameters

In this section we define the parameters—for quantitatively describing load, resources, and performance—on which the first scalability evaluation will be based. The analysis of the gained results is expected to give indications on suitable adjustments for use in further experiments. Hence the procedure outlined above can be viewed as one iteration of an iterative scalability evaluation, in which each iteration helps in refining the underlying setting towards producing more significant conclusions. We start with some simplifying assumptions (which are subject of gradual release):

- there are no subspaces,
- all operations are permitted by the security manager,
- neither errors nor exceptions occur,
- transactions are not used.

7.2.1 Load

The load parameters are based on a kernel[1] (the number of kernels is seen as a resource rather than load parameter). Initial value ranges to be measured are given. Not all combination of values are necessarily feasible (e.g., the combination of all maximal values may be beyond the capability of the prototype, whereas setting only a subset of the values to the maximum would be tolerable).

numberOfSpaces the number of spaces managed by a kernel `[1..100]`

numberOfSpacesRd the number of spaces actually read (without URL) `[1..10]`

numberOfTriplesStored the number of triples stored in a space `[100..100000]`

tripleSizeStored the average size of triples data stored `[100B..1KB]`

numberOfClients the number of clients simultaneously issuing requests to a kernel `[1..1000]`

requestFrequency the frequency of requests (per second, per client) `[0,01..10]`

numberOfTriplesTransferred the number of triples transferred between kernel and client `[100..1000]`

tripleSizeTransfered the average size of triples transferred `[100B..1KB]`

APIFunction as a non-quantitative parameter, load is also characterized by the kind of TS-API function issued.

7.2.2 Resources

The parameters to quantify resources are

numberOfKernels the number of kernels `[1..100]`

numberOfManagingKernels the number of kernels managing a space (assuming the distribution of spaces onto multiple kernels) `[1..10]`

numberOfMachines the number of physical machines on which a kernel is executed (assuming kernel components running on different machines) `[1..10]`

numberOfInstances_component the number of instances of a component (assuming multiple instances of components) `[1..10]`

7.2.3 Performance

The performance parameter is time, indeed in different variants.

responseTime the period of time between calling a TS API function and control returning to the client program (in case of non blocking operations)

[1] more precisely, a kernel instance

modificationTime the period of time between calling a TS API function and the effect on persistent storage (in case of operations which change the contents of a space)

notificationTime the period of time between changing the contents of a space and the completed receipt of data by the callback (in case of subscriptions)

7.3 Evaluation Environments

In the following, on overview of potential testbeds for the performance and scalability evaluation of the TripCom prototype is given. Table 7.1 summarizes the main features of these environments.

7.3.1 PlanetLab

PlanetLab[2] is a geographically distributed overlay network designed to support the deployment and evaluation of planetary-scale network services. It enables a large research community to share its infrastructure through providing distributed virtualization, i.e., each service runs in an isolated *slice* of PlanetLab's global resources. In addition, in order to support competition among multiple network services, PlanetLab follows the principle of *unbundled management*, i.e., the operating system running on each node is decoupled from the network-wide services that define PlanetLab.

A slice is a set of allocated resources distributed across PlanetLab. Usually a slice represents shell access to a number of PlanetLab nodes. The general scheme of the use of slices is the following. Slices are created (1), and assigned to users (2). After being assigned to a slice, a user may then assign nodes to it (3). After nodes have been assigned to a slice, virtual servers for that slice are created on each of the assigned nodes (4). An important property of slices is their limited lifetime; they must be periodically renewed to remain valid.

As a prerequisite of using PlanetLab, an institution must be member of the PlanetLab Consortium. For academic members, the use of up to 10 slices is free of charge.

7.3.2 Amazon Elastic Compute Cloud

The Amazon Elastic Compute Cloud (EC2)[3] is a Web service which provides to the user a custom application environment on a set of distributed machines. The EC2 is a virtual computing environment. The number of machines, the application executed, and the network access (e.g., the firewall settings) can be configured dynamically through a Web service interface.

The user's application environment within the computing cloud is represented as an *Amazon Machine Image* (AMI). An AMI contains all the applications, libraries, data and configuration settings required for execution. In order to allow for the dynamic loading of AMIs via the Web service interface, they have to be uploaded to Amazon's *Simple Storage Service* (S3). Multiple AMIs with different configurations and applications can be uploaded to the S3. The user can start, monitor, and terminate as much instances of these interfaces as needed.

[2] `https://www.planet-lab.org`
[3] `aws.amazon.com/ec2`

As a prerequisite of using the Amazon Elastic Compute Cloud, an institution must hold a (free) amazon.com account.

7.3.3 TUW Campus Grid

The Technical University of Vienna has about 300 computers in 14 computer "internet" rooms for students.[4] These computers can be used as a cluster when the computer rooms are closed (during the holidays and in the night[5]). The machines can be accessed as one grid using Condor[6], or every single machine can be accessed individually using a SSH-shell. The campus grid is controlled by a so called grid master. The network at TUW is automatically reconfigured to boot the machines with a special boot image which is located at the grid master when the computer rooms are closed. Every member of TUW can apply for an account. The grid master machine and all the hosts in the grid can be accessed with this account and the jobs which shall be executed can be planed and prepared. The actual computation of the jobs can be scheduled to be executed during the night.

7.3.4 Selection

Due to its advantages as listed in Table 7.1, to mention in particular the flexible and highly automated handling due to the virtual machine concept, eventually the Amazon Elastic Compute Cloud has been chosen as the platform for the scalability evaluation of TripCom.

7.4 Evaluation Results

The performance data generated by the measurement programs will be collected in a statistical software tool, which also allows for the graphical presentation of the data (plotting various dimensions of load, resources, performance against each other to derive characteristic curves), thereby facilitating the analysis and interpretation of the results. The conclusion on the scalability of TripCom will be reported.

[4] http://www.zid.tuwien.ac.at/student/internet_raeume
[5] http://www.zid.tuwien.ac.at/zidline/zl14/grid.html
[6] http://www.cs.wisc.edu/condor

	Amazon EC2	Planetlab	TU Vienna campus grid
number of machines	20 per account; can ask for more, target in order to stay within budget: 50 machines	842 nodes (21.02.2008) in 416 sites	approx. 250
management/ deployment of machines	virtual machine images (containing applications, libraries, configurations, ...) can be created and loaded as needed	shell acces to nodes which are assigned to a slice	ssh access to all machines; or via Condor queue system
requirements	Amazon account	member of Planetlab	member of TU Vienna
machines externally visible?	yes	not sure about direct slice accessibility	no
price	\$0.10/hour and machine \$0.10/GB data into the cloud \$0.18/GB data from the cloud cost calculator[a]	for academic members: free, can have 10 slices	free
master copy automatically distributable?	yes (virtual machines, including os, programs and configurations)	need to install every machine separately, but tools exist to execute same command on a number of nodes simultaneously	yes: machines share common home directory
scriptable deployment	yes, Amazon provides libraries/API for management of images	yes (see above)	yes
types of machines	listed[b]	unknown, varying resources (esp. CPU) not guaranteed	varying; PC's of medium capacity
operating system	can be chosen (virtual machine)	Linux in a virtual machine	Linux (kernel 2.6.18)
restrictions	no	no extensive use of resources allowed, nodes can be reinstalled or rebooted without warning, no performance tests allowed!	time (night, weekends, holidays)
exclusive use of machines	yes	no	mostly yes; may require some arrangement with others
machine specific directory/file names	yes	?	yes (scriptable)

Table 7.1: Testbeds for prototype evaluation

[a] http://calculator.s3.amazonaws.com/calc5.html?

[b] http://www.amazon.com/b/ref=sc_fe_c_0_201590011_2?ie=UTF8\&node=370375011\&no=201590011\&me=A36L942TSJ2AJA

8 Conclusions

In this work, the result of major tasks carried out on the levels of software design, architecture and implementation in order to ensure proper scalability (i.e. web scale) of Triple Space is presented. By following six steps outlined in the introduction, we defined the overall objectives of the newly introduced scalability task, specified core concepts and definitions regarding scalability and its specific meaning to our project. As a first step towards achieving our goal, we identified *scalability factors* and their mutual influence amongst each other, e.g. the impossibility to achieve complete results but still guaranteeing very low latency. We solve this issue by introducing so called *configurations*, means to describe a particular realization of a set of scalability factors. Three configurations where identified, covering the full spectrum of functionality of Triple Space, while at the same providing Quality of Services required by the TripCom use cases. The second major contribution of this work is the analysis of approaches to achieve large scale given the identified scalability factors, followed by our design of the deployment architecture of Triple Space. In this part, we emphasize the need for a supporting infrastructure that provides support for distributed indexing, system dynamism, replication, backup, scalability, monitoring and security. For the reason of better scalability when compared to centralized and de-centralized approaches and better support for monitoring when compared to DHTs, a semi-decentralized system design was chosen.

As the last step on the proposed agenda, we presented a setup and measures to test our system in a large, distributed environment with various workloads, participating peers and clients and prove, that our design provides true web scale.

References

[1] K. Aberer, Ph. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: A Self-organizing Structured P2P System. *SIGMOD Record*, 32(3), September 2003.

[2] K. Aberer, Ph. Cudré-Mauroux, M. Hauswirth, and T. van Pelt. Gridvine: Building internet-scale semantic overlay networks. In *3rd Int'l Semantic Web Conference*, November 2004.

[3] St. Androutsellis-Theotokis and D. Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.

[4] M. Cai and M. Frank. RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network. In *13th Int'l Conference on World Wide Web*, 2004.

[5] D.F. Carr. How Google Works. Baseline Web Magazine, www.baselinemag.com, July 2006.

[6] J.J. Carroll, Ch. Bizer, P. Hayes, and P. Stickler. Named Graphs, Provenance and Trust. In *14th Int'l Conference on World Wide Web*, pages 613–622, May 2005.

[7] R. Cyganiak. A relational algebra for SPARQL. Technical Report HPL-2005-170, Hewlett Packard Laboratories, September 28 2005.

[8] D. de Francisco, D. Martin, Th. Scheibler, D. Wutke, A. Harth, M. Murth, and E. Simperl. Tripcom requirements analysis and architecture profile for eai applications. Technical report, March 2007.

[9] A. Deshpande, Z.G. Ives, and V. Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.

[10] D. Fensel. *Problem-Solving Methods: Understanding, Description, Development, and Reuse*. Springer Verlag, 2000.

[11] D. Fensel. Triple-Space Computing: Semantic Web Services Based on Persistent Publication of Information. In *IFIP Int'l Conf. on Intelligence in Communication Systems*, pages 43–53, November 2004.

[12] D. Fensel. Computing for the World: Incomplete, incorrect but requested! *IEEE Intelligent Systems*, 22(6), November/December 2007.

[13] D. Fensel, E. Motta, St. Decker, and Z. Zdrahal. The use of Ontologies for Specifying Tasks and Problem Solving Methods: A Case Study. In *10th European Workshop on Knowledge Acquisition, Modelling, and Management*, 1997.

[14] A. Haller, E. Cimpian, A. Mocan, E. Oren, and Ch. Bussler. WSMX - A Semantic Service-Oriented Architecture. In *IEEE Int'l Conference on Web Services*, 2005.

[15] A. Kiryakov. Measurable Targets for Scalable Reasoning. Technical report, Ontotext, November 2007.

[16] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Survey*, 32(4):422–469, 2000.

[17] R. Krummenacher, E. Simperl, and D. Fensel. Towards Scalable Information Spaces. In *Workshop on new forms of reasoning for the Semantic Web: scaleable, tolerant and dynamic, ISWC*, Nov. 2007.

[18] R. Krummenacher, E. Simperl, V. Momtchev, L.J.B. Nixon, and O. Shafiq. Specification of the Triple Space Ontology. TripCom Deliverable D2.2, March 2007.

[19] R. Krummenacher, E. Simperl, L.J.B. Nixon, D. Cerizza, and E. Della Valle. Enabling the European Patient Summary Through Triplespaces. In *20th IEEE Int'l Symp. on Computer-Based Medical Systems*, June 2007.

[20] J.C. Laprie. Dependable Computing and Fault Tolerance: Concepts and Terminology. In *15th IEEE Int'l Symposium on Fault-Tolerant Computing*, 1985.

[21] V. Momtchev and A. Kiryakov. Specification of the Store Architecture and Interfaces. TripCom Deliverable D1.2, September 2006.

[22] E. Motta. *Reusable Components for Knowledge Modelling*, volume 53 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 1999.

[23] E. Motta and Z. Zdrahal. Parametric Design Problem Solving. In *10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 9.1–9.20, November 1996.

[24] M. Murth, G. Joskowicz, e. Kuhn, D. Cerizza, D. Cerri, D. de Francisco, A. Ghioni, R. Krummenacher, D. Martin, L.J.B. Nixon, N. Sanchez, B. Sapkota, O. Shafiq, and D. Wutke. Triple Space Reference Architecture. TripCom Deliverable D6.2, March 2007.

[25] L.J.B. Nixon, D. Martin, D. Wutke, M. Murth, E. Simperl, R. Krummenacher, B. Sapkota, Z. Zhou, H. Moritsch, O. Shafiq, G. Toro del Valle, D. Cerri, and V. Momtchev. Platform API Specification for Interaction Between All Components. TripCom Deliverable D6.3, March 2008.

[26] L.J.B. Nixon, Ph. Obermeier, O. Shafiq, J. Saarela, and H. Munoz. Semantic matching in distributed spaces. Technical report, March 2008.

[27] L.J.B. Nixon, E. Simperl, R. Krummenacher, F. Martin-Recuerda, V. Momtchev, M. Murth, G. Joskowicz, and e. Kuhn. Specification and implementation of a semantic linda model. Technical report, March 2007.

[28] L.J.B. Nixon, K. Teymourian, R. Krummenacher, and V. Momtchev. Semantic Clustering and Self-Organization in Triple Space. TripCom Deliverable D2.4, March 2008.

[29] B. Sapkota, D. Foxvog, D. Wutke, D. Martin, M. Murth, O. Shafiq, A. Turati, E. Della Valle, N. Sanchez, and J. Kopecky. Architectural integration of triple spaces with web service infrastructures. Technical report, Tripcom, 2007.

[30] O. Shafiq, D. Cerizza, J. Kopecky, D. Martin, M. Murth, B. Sapkota, G. Toro del Valle, A. Turati, and D. Wutke. Tripcom grounding for semantic web services. Technical report, March 2008.

[31] O. Shafiq, R. Krummenacher, F. Martin-Recuerda, Y. Ding, and D. Fensel. Triple Space Computing middleware for Semantic Web Services. In *Workshop on Middleware for Web Services, EDOC*, 2006.

[32] E. Simperl, L.J.B. Nixon, R. Krummenacher, V. Momtchev, and H. Munoz. Representing rdf semantics in tuples. TripCom Deliverable D2.1, March 2007.

[33] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2005.