Towards Scalable Information Spaces

Reto Krummenacher, Elena Simperl, and Dieter Fensel

Digital Enterprise Research Institute, University of Innsbruck, Austria
firstname.lastname@deri.at

Abstract. Recent work in the field of middleware technology proposes semantic spaces as a tool for coping with the *scalability*, *heterogeneity* and *dynamism* issues of large scale distributed IT environments. In this paper we discuss the impact that functional and non-functional requirements have on the scalability of a semantic space installation. Based on this analysis we elaborate on the application scenarios in which semantic information spaces are expected to be a feasible middleware solution for Web scale communication and coordination.[1]

1 Introduction

Service orientation is rapidly becoming the dominant computing paradigm. However, current technologies are still restricted in their application context to the in-house solution of companies' internal problems. While in theory the potential of service oriented computing, as means to abstract from underlying hardware and software layers, is widely acknowledged, its success depends on resolving the fundamental challenges that arise if the current technologies are to be used in open, heterogeneous, distributed, and fast changing environments like the Web.

Novel middleware solutions that enable the communication and coordination of billions of services are an essential building block for bringing service orientation to Web scale. Recent work in the field of middleware promotes semantic spaces as potential solution. One important characteristic of semantic spaces is the persistent publication of data that ensures the information and messages to outlive their producers and consumers, which in turn decouples the interacting services in time and reference. Semantic spaces are set at the intersection of Linda-like tuplespace computing [6] and the Semantic Web – our approach is referred to as Triple Space Computing [3, 5, 7, 11]. Triple Space Computing (TSC) provides a set of interaction primitives that allow the publishing and template-based retrieval of RDF triples to respectively from addressable triplespaces. A triplespace is a bag or container that implicitly groups published RDF triples. Detailed information about the Triple Space organizational models is published in [11], while further insights about the interaction primitives are given in Section 2.

Before looking at the functional and non-functional factors that influence the scalability of semantic space installations we shortly present our assumptions with respect to Triple Space Computing: openness, dynamism, distributedness, decentralization and scalability at Web scale.

[1] This work is supported by the project TripCom (IST-4-027324-STP, www.tripcom.org)

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.
Dynamism: Hosts and information sources may intentionally or not go offline, spaces may be created or only temporally be put in place.
Distributedness: 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 even be shared by multiple hosts and decentralization is emphasized (Fig. 1).
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 (distributedness).

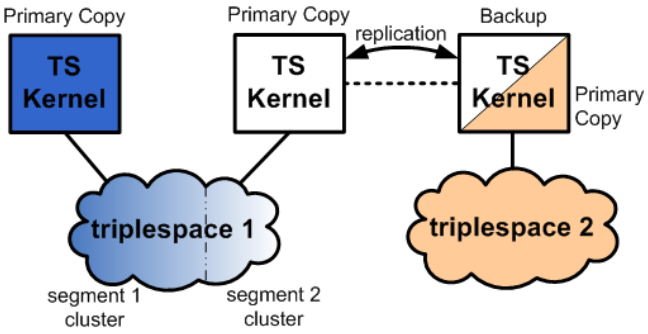

Fig. 1. The decentralization of Triple Space with respect to hosts (TS Kernels) and spaces.

The next section considers the functional and non-functional aspects that influence the scalability of the Triple Space. Thereafter, in Section 3, we present the main scalability trade-offs that lead to a discussion of the scalability levels that are offered by the Triple Space in Section 4. The paper is concluded with Section 5.

2 Functional and Non-Functional Scalability Factors

There are two dimensions that influence scalability: on the one hand there is the perceived functional service that Triple Space delivers, while on the other there are a number of non-functional properties that impact the overall scalability. The non-functional properties do not define the services of the middleware, but rather describe how the services behave - invisible to the user. Such non-functional aspects are for example availability, fault-tolerance, completeness, durability, response time, and also security.

2.1 Functional Factors

In this paper we only present the core functionality of the Triple Space – as detailed by the TripCom project – that enables the publication and retrieval of semantic data.

Further interaction primitives for invalidation or deletion of data, or for notification services are based upon these and treated analogously.

Publication: The core operation for publication atomically writes a single ***triple*** into the ***space***. The operation makes no guarantee if and when the triple will be available and persistently stored in the space (unordered semantics). The client is immediately free to perform further activities. The client has to provide a resolvable URL which identifies the ***space***. An extended version provides the possibility to atomically write a ***set of triples*** into the ***space***.

Retrieval: The core retrieval operation returns one match of a given ***template*** which is a single triple pattern (e.g <:tripcom ?p ?o>) that is discovered in the ***space***. In order to pay tribute to the graph structure of RDF data, the match may be a set of triples, e.g. a Concise Bounded Description [12].

```
rd(Template t, URL space, Time timeout):Set<Triple> s
```

The operation makes no guarantee as to when the match would be returned to the client. An extended version with the same signature considers more expressive templates (RDF queries, rules) for which the syntax and the actual set of triples returned is determined by the matching rules. A ***timeout*** is accepted to indicate a temporal bound; if no match has been found by the timeout period, an empty set is returned. This does not make any statement regarding the existence of a match in the space. In case the client does not specify a space URL, the middleware must autonomously determine the relevant sources (cf. Section 4).

Orthogonally to the operational functions there are two further dimensions that influence the functionality of the Triple Space: inference, and the expressivity or query complexity resulting from enhanced template matching algorithms.

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. Considering for example invalidation of explicit data, or knowledge respectively: 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 make sure 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 in Triple Space is based on simple triple patterns. Matching approaches based on triple patterns can scalably be realized for large scale open systems like various semantic P2P-implementations have shown [1, 2].[2] The target applications of Triple Space demand however more complex

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

semantic query languages, or even rule-based query resolution; e.g. [9]. These more complex matching algorithms provide more functionality at the price of a more complex implementation, in particular in distributed settings, as we will discuss later in this paper.

2.2 Non-Functional Factors

In order to better address the non-functional properties we first outline a set of core measures that are necessary – not imperatively sufficient – to realize the non-functional properties for a triplespace under the above given assumptions.

Redundancy: Redundancy is defined as "the provision of additional or duplicate systems, equipment, etc., that function in case an operating part or system fails".[3] First, there might be redundant components within a TS Kernel that ensure the expected functionality.[4] 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.

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 (cf. Fig. 1):

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. The data retrieved from different kernels might differ, as complete access to all data cannot be guaranteed – however, this does not violate the core expectations of Triple Space retrieval, as outlined earlier in this section.

Having the available techniques in place we now present the non-functional properties and depict how they can be addressed.

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

[3] http://www.dictionary.com

[4] A kernel is the sum of components, which as a whole provide the TS functionality of a host.

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:

$$A = \frac{E[uptime]}{E[uptime] + E[downtime]} \tag{1}$$

Technical measure: Load balancing, i.e. the distribution of responsibilities, is a very promising measure to increase the operational continuity of a system.

Reliability: According to IEEE reliability is "the ability of a system or component to perform its required functions under stated conditions for a specified period of time".[5]

Technical measure: A popular measure against a lack of reliability is redundancy of the internal processes that control each other in a two-out-of-two partnership. In cases where reliability is given through availability (e.g. documents are available and hence assumed reliable), replication is often used instead of redundancy - with the same effect. The risk with replication is the loss of consistency, as it is not ensured that multiple copies of the same resource reflect the same state and content.

Fault-tolerance: Fault-tolerance allows a system to continue operating properly in the event of a failure of one of more of its components.

Technical measures: An important factor with respect to fault-tolerance is the ability to recover. However, more important is to avoid system failures. This is first of all achieved by keeping the system complexity low, and second through redundancy:

- 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.

Redundancy and load balancing (may) however increase the complexity - thus we have a first trade-off. Systems with significant fault-tolerance requirements rely on so-called two-out-of-three solutions, where the component detecting a failure has time to recover without negatively influencing the overall performance of the system.

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

Technical measures: Security measures prevent potential intruders from accessing the system or the managed information through login protection, encryption or digital signatures. The measures are orthogonal to the other technologies presented, just as the property is orthogonal to the other non-functional aspects.

Safety: Safety is about failures caused by system internal components and procedures. Safety is only mentioned here for reasons of completeness; according to [10] 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.

[5] IEEE Reliability Society: http://www.ieee.org/portal/site/relsoc/

Completeness: This property refers to the amount 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. In information retrieval this property is called recall: the proportion of relevant documents that are retrieved, out of all relevant documents available (2).

Technical measure: In the context of information spaces *no* distribution of data is a necessary and sufficient measure - assuming the well-functioning of the kernel's query engine.

$$R = \frac{relevant \cap retrieved}{relevant} \qquad (2) \qquad P = \frac{relevant \cap retrieved}{retrieved} \qquad (3)$$

Correctness: In many cases it is trivial to achieve a recall of 100% by returning all data in response to any query.[6] Therefore recall is paired with precision, the number of non-relevant document returned - i.e. the correctness of retrieval. In information retrieval the correctness indicator is called precision and refers to the ratio of false data that was retrieved (3). In semantic space computing these false positives reflect the non-matching statements that are returned.

Technical measures: A correct matching algorithm delivers a solution to correctness - assuming availability of a sufficiently complete set of statements. Otherwise the matching algorithm might draw false conclusions and return false positives.

Consistency: The consistency issue was shortly addressed in the discussion about reliability. Processing copies of data that are stored at different kernels, which are potentially not globally known, may result in contradicting statements. Note that the TripCom model does not fully guarantee consistency, as users cannot be hindered from publishing contradicting facts. It is not seen as a task of the middleware to manipulate user data in order to achieve consistency. However, it is important that the system does not add inconsistencies.

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

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 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 TSC by enabling "Semantic Web services based on persistent publication of information" [3].

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

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. Performance is easiest achieved by only relying

[6] Obviously not as trivial in distributed settings, where some providers might not be known.

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 measure: A first solutions is thus to limit the system size and to keep the complexity low. Furthermore, load balancing is a appropriate technique to decrease the response time by reducing congestions at kernels and kernel components.

Table 1. Relationships between core TS functionalities and non-functional properties

	Publication	Retrieval
Availability	Y	Y
Reliability	Y	Y
Fault-tolerance	Y	Y
Security	Y	Y
Completeness	N	Y
Correctness	N	Y
Consistency	N	Y
Durability	Y	N
Performance (Response Time)	Y	Y

As summery, Table 1 shows the relationship between the core functionalities of Triple Space and the non-functional properties introduced.

3 Scalability Trade-Offs

The trade-offs one has to be aware of when designing or implementing a scalable semantic space middleware at Web dimensions are between non-functional properties under the clear influence of the desired functionals. After having the necessary definitions in place, we will now concentrate on the trade-offs that determine the scalability-driven structural and behavioral choices of a kernel instance. As fundamental observation we state that the richer functionality the middleware provides the less can be guaranteed for the non-functional aspects and for the scalability of the overall infrastructure in particular.

The indicative set of determined trade-offs between non-functional properties is given in Table 2. A first finding shows that there is a general trade-off between dependability, i.e. trustworthiness of the middleware and the quality of response with respect to the handling of data. Distribution of responsibilities and data increases the dependability of the middleware, while it decreases - depending on the approach chosen - the completeness, consistency and correctness of the stored and returned data.

Other apparent trade-offs are related to the performance property. Hidden computations like consistency checking, distributed writing of data, or the querying of distributed and a priori not known sources adds to the latency. This becomes particularly evident when considering a Web scale implementation, where a client can query the

Table 2. Trade-offs between non-functional properties

Correctness ⇔ Completeness	The well known trade-off between precision and recall.
Completeness ⇔ Availability	Availability can be achieved by load balancing which in turn might rely on data distribution.
Consistency ⇔ Availability	Availability can be achieved by load balancing which in turn might rely on data replication.
Consistency ⇔ Fault-tolerance	Replication and redundancy are measures for fault-tolerance.
Reliability ⇔ Response Time	Reliability based on redundant execution introduces overhead in time requirements.
Completeness ⇔ Response Time	Completeness at larger scale requires discovery and querying of distributed data sources.
Durability ⇔ Response Time	Persistent writing at remote locations increases the latency of out calls.

whole Triple Space without indicating a source space (rd without specification of a space URL). This case is further discussed in Section 4.

In addition to the trade-offs shown in Table 2 we should not neglect the influence of query complexity and inferred data on the non-functional properties. The two functionalities are clearly related: query resolution, depending on the complexity of the language, is a particular type of reasoning - just as inference is based on reasoning.

As mentioned in Section 2, simple triple pattern matching is scalable in P2P networks. Due to the fact that such projects only considers isolated triples and not their semantics and connectivity in graphs, it is possible to match the data by any of the three fields of the triple. Hence, they hash the three URIs separately and distribute the triple accordingly. More complex query algorithms however depend on the interpretation of the data and consequently rely on a more complete view of the available data. This requirement becomes particularly influential when query resolution and inference is performed in decentralized and distributed environments - like assumed for semantic information spaces. Therefore there is a clear relationship between the complexity of the reasoning-based functionalities and the non-functional properties that are addressed by distribution of data. This trade-off manifests in retrieval without spaces URL, where reasoning over distributed sources might become a requirement.

4 Levels of Triple Space Scalability

The previous sections have shown that the scalability issue is mostly apparent in the context of the retrieval functionality of Triple Space. On the one hand scalability is more tangible at retrieval time also for users, and on the other the Semantic Web influences (e.g. inference) are primarily detectable 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 paper 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 [4]. 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.

5 Conclusion

In the scope of the TripCom project scalability is a major concern, as the developed semantic space middleware shall offer a communication and coordination platform for large scale open systems. This paper presented the relevant functional and non-functional properties that influence the scalability of a semantic information space implementation. Furthermore we presented scalability trade-offs that will help in accomplishing the right structural and behavioral choices with respect to performance, scalability and other non-functional aspects. The formal modeling of the trade-offs and the resulting choices is ongoing work – first ideas for ontology-driven management were already presented in [8].

The discussion of Triple Space scalability levels has shown that the semantic space middleware provides at least local scalability by naturally grouping related data and users in spaces. However, as soon as the chosen installation moves away from having a single space as source of information the problem of completeness manifests itself again. The main trade-off of semantic space computing is thus not surprisingly relating dependability (enabled by load balancing, failure-tolerance and redundancy that are at least partly implemented by distribution-based techniques) and completeness. Consequently, even though TSC provides some basic measures for the realization of scalable Semantic Web middleware, it too might require novel techniques that allow for large scale reasoning or more realistically, reasoning with incomplete data.

References

1. K. Aberer, P. Cudré-Mauroux, M. Hauswirth, and T. van Pelt. GridVine: Building Internet-Scale Semantic Overlay Networks. In *3rd Int'l Semantic Web Conference*, November 2004.
2. M. Cai and M. Frank. RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network. In *13th International Conference on World Wide Web*, 2004.
3. D. Fensel. Triple-Space Computing: Semantic Web Services Based on Persistent Publication of Information. In *IFIP Int'l Conf. on Intelligence in Communication Systems*, Nov. 2004.
4. D. Fensel. Computing for the World: Incomplete, incorrect but requested! *IEEE Intelligent Systems*, 22(6), November/December 2007.
5. D. Fensel, R. Krummenacher, O. Shafiq, E. Kuehn, J. Riemer, Y. Ding, and B. Draxler. TSC - Triple Space Computing. *e&i Elektrotechnik und Informationstechnik*, 124(1/2), Feb. 2007.
6. D. Gelernter. Generative Communication in Linda. *ACM Trans. Prog. Lang. Syst.*, 7(1):80–112, 1985.
7. R. Krummenacher, M. Hepp, A. Polleres, C. Bussler, and D. Fensel. WWW or What is Wrong with Web services. In *3rd European Conf. on Web Services*, November 2005.
8. R. Krummenacher, E. Simperl, and D. Fensel. An Ontology-Driven Approach To Reflective Middleware. In *IEEE/WIC/ACM Int'l Conference on Web Intelligence*, November 2007.
9. R. Krummenacher, E. Simperl, L. 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.
10. J. Laprie. Dependable Computing and Fault Tolerance: Concepts and Terminology. In *15th IEEE Int'l Symposium on Fault-Tolerant Computing*, 1985.
11. E. Simperl, R. Krummenacher, and L. Nixon. A Coordination Model for Triplespace Computing. In *9th Int'l Conference on Coordination Models and Languages*, June 2007.
12. P. Stickler. CBD - Concise Bounded Description. W3C Member Submission, June 2005.