Introducing the Concept of Customizable Structured Spaces for Agent Coordination in the Production Automation Domain*

eva Kühn, Richard Mordinyi, László Keszthelyi, Christian Schreiber
Space-based Computing Group
Vienna University of Technology
Argentinierstr. 8
1040 Vienna, Austria
eva,richard,laszlo,cs@complang.tuwien.ac.at

ABSTRACT

Tuple spaces are a common platform for the coordination of agents. In the past years there have been several approaches of improving the concept of coordination via the shared space. However, some of those concepts, like the Programmable Matching Engine, were primarily concentrating on retrieving tuples from the space with improved query techniques.

In this paper, we propose the concept of structured spaces, so called Space Containers, which allow to store tuples in a customizable structured way. The concept of a Space Container allows a) to distinguish between the data needed for coordination purposes only and the payload, b) enables an explicitly structured way of storage and retrieval of the stored data, and c) the realization of more complex coordination patterns. The benefits of the proposed approach are a) less complex agent implementations, and b) the possibility of an efficient implementation of coordination issues.

We describe the architecture of the proposed approach, explain the benefits of it by means of a scenario from the production automation domain and show evaluation results.

Categories and Subject Descriptors

I.2 [**ARTIFICIAL INTELLIGENCE**]: Distributed Artificial Intelligence

General Terms

Design, Measurement, Performance

Keywords

Linda, Tuple Spaces, Coordination, Coordination Pattern, Multi-Agent Systems, Production Automation

1. INTRODUCTION

In [9], Gelernter and Carriero promote a clear separation between the computation model, used to express the computational requirements of an algorithm, and the coordination model, used to express the communication and synchronization requirements. They explain that these two aspects of a system's construction may either be embodied in a single language or, as they prefer, in two separate, specialized languages. The latter approach has been realized in the Linda coordination language based on the tuple space model [8].

Tuple Spaces [4] are an accepted platform for the purpose of coordination of e.g. software agents [1]. In the recent years, several frameworks have been developed in order to coordinate agents more efficiently. In a nutshell, they focused on e.g. mobile agents support [19, 2], introduction of subspaces [5], GRID environments [12, 14, 3], or improving the tuple matching mechanisms (Section 2.1) in order to increase the number of possible tuples matching a specific query.

In general, they refer to the same principles as described in [8]. There, briefly, coordination is described as a process of writing tuples and reading of tuples matching a specific pattern. The principle of having an unstructured tuple space with random, non-deterministic tuple access implies that the usage of even slightly more complex coordination patterns with ordering characteristics, like FIFO or LIFO, cannot be handled by the coordination framework completely and requires therefore additional actions of coordination management taken by the participating agents. This on the other hand would lead to the fact of an unclear separation between computation model and the coordination model.

In this paper, we propose the concept of structured tuple spaces by means of so called Space Containers [16]. A Space Container allows the storage of entries (Section 5) in a customizable structured, ordered way. Entries are data structures of any type, and not restricted to tuples only. The ordered, structured form of the space is achieved by specific customizable Coordinators, an inherent component of a Space Container. By distinguishing explicitly between data needed for coordination purposes and the payload itself, a Space Container is capable of having its own specific view on the space. Those coordination data is used by the Coordinators to form their specific order of the entries, and so representing the required form of coordination. Since it is a specific, customized view on the entries in the space, these views can be implemented in an optimized way regarding the semantics and aim of the specific coordination pattern,

*The project is partly funded by TripCom (IST-4-027324-STP project, http://www.tripcom.org).

Cite as: Introducing the Concept of Customizable Structured Spaces for Agent Coordination in the Production Automation Domain, Eva Kühn, Richard Mordinyi, László Keszthelyi, Christian Schreiber, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 625–632

and thereby helping the agent to work efficiently.

The benefits of the concept are a) less complex agent implementations since the separation between the computation model and the coordination model is given, and b) the implementation of efficient coordination sequences since the Coordinator has been explicitly designed to support the required form of coordination.

The remainder of this paper is structured as the following: section 2 summarizes related work, section 3 defines the research questions, section 4 pictures the use case, section 5 describes the concept and the architecture of Space Containers, while section 6 discusses the evaluation results. Finally section 7 concludes the paper and proposes further work.

2. RELATED WORK

This section summarizes related work on the usage of tuple spaces in the production automation domain and on tuple retrieval approaches in tuple space implementations.

2.1 Tuple retrieval in Spaces

The Linda coordination model [8], developed in the mid-1980's by David Gelernter at Yale University, is the originator of the "space based system". It describes the usage of a logically shared memory, called *tuple space*, together with a handful operations (`out`, `in`, `rd`, `eval`) as a communication mechanism for parallel and distributed processes. Principally, the tuple space is a bag containing tuples with non-deterministic `rd` and `in` operation access. A tuple is buildup of fields containing a value and its type, whereby unassigned are not permitted, e.g. a tuple with three fields is < "point", 12, 67>, where "point" is of type string and 12 resp. 67 are of type integer.

The defined operations allow placing tuples into (`out`) the space and querying tuples from the space (`rd` and `in`). The difference between `rd` and `in` is that `rd` only returns a copy of the tuple, whereas `in` removes it as well from the tuple space. Both operations return only a single tuple and will block until a matching tuple is available in the tuple space. There are also non-blocking versions of the `rd` and `in` operation, called `rdp` and `inp`, which return an indication of failure instead of blocking, when no matching tuple is found [24].

The Linda model requires the specification of a tuple as an argument for both query operations. In such a case, the tuple is called template that allows the usage of a wildcard as the field's value. A wildcard declares only the type of the seeked field, but not its value, e.g. the operation rd("point", ?x, ?y) would return a tuple, matching the size, the type of the fields and the string "point". A tuple containing wildcards is called an anti-tuple. If a tuple is found, which matches the anti-tuple, the wildcards are replaced by the value of the corresponding fields. The non-deterministic `rd` and `in` operation semantics comes from the fact that in case of several matching tuples a random one is chosen.

Implementations that support the above mentioned exact tuple matching are: Blossom [22], JavaSpaces [7], LIME [19], MARS [2] and TuCSon [5]. Although both MARS and TuCSoN enable the modification of the operations' semantics by adding so called *reactions*, they can not influence the way how tuples are queried. JavaSpaces adds subtype matching to the exact tuple matching mechanism to query objects from the space.

The drawback of exact tuple matching is that all collaborating processes must be aware of the tuple's signature they use for information exchange. Hence, there are several tuple space implementations that offer additional queries mechanisms, such as TSpaces [25, 18], XMLSpaces.Net [21] and eLinda [23]. TSpaces offers the possibility to query tuples by named fields or by specifying only the field's index and a value or wildcard. Furthermore, TSpaces allows the definition of custom queries by introducing the concept of factories and handlers. Both TSpaces and XMLSpaces.Net support the use of XML-documents in tuple fields and therefore enable the use of several XML query languages such as XQL or XPath. In addition, XML-Spaces.Net uses an XML-document like structuring for its space, which allows the utilization of sophisticated XML queries on the space. eLinda [23] enables the usage of more flexible queries, via its Programmable Matching Engine (PME), such as maximum or range queries. Beside these queries the PME also provides *aggregated operations* that allow the summary or aggregation of information from a number of tuples, returning the result as a single tuple. The PME allows, like TSpaces with its concept of custom factories and handlers, the simple definition of custom matchers [23].

It can be concluded, that all previously introduced tuple space implementations have in common that the stored tuples have no ordering. Furthermore, they do not guarantee which tuple is returned by a query, it may happen that due to the non-deterministic semantics of the Linda operations a tuple is never returned although it would match the query.

2.2 Multi-Agent Systems in Production Automation

Hierarchical or centralized structures are mostly used by present manufacturing systems. Therefore, they are inflexible and not able to react effectively and efficiently to unforeseen disturbances. The decentralized control architecture concept of Multi Agent Systems (MAS) [11] is well suited to address these kinds of problems because they provide modularity, improved flexibility and robustness against failures [15].

The coordination of agents is still a key issue in Multi-Agent Systems, and a basic approach is to use tuple space based coordination frameworks. Furthermore, the tuple space based architecture can be used for coordinating test agents in a Multi-Agent test framework as proposed in [26]. They introduce the usage of "reactive" tuple space systems, which allow the definition of "reactions" that are triggered by specific event occurrences. A reactive tuple space systems realizes a hybrid coordination model that combines the cleanness and elegance of data-driven coordination models and the flexibility and the power of control-driven ones [1].

3. RESEARCH QUESTIONS

In this paper, we propose the concept of structured spaces, so called Space Containers, which allow to store tuples in a customizable structured way. Based on the limitations of the traditional tuple space model with respect to coordination forms requiring ordered tuples and on recent projects with industry partners from the production automation, we derived the following research questions:

R.1 - Concept of Space Containers: Investigate a) the advantages and limitations of the Space Container supported form of coordination capabilities, and b) whether the proposed concept allows reducing the complexity of agent

implementations while supporting at the same time more complex coordination patterns. What are the major differences between the structured space approach and the traditional tuple based form of coordination?

R.2 - Efficient implementation of coordination patterns: Investigate to what extent the usage of Coordinators and Selectors help to coordinate efficiently. Does the concept of separating coordination data and payload contribute to an overall efficiency improvement?

For investigating these research issues, we gathered requirements from a set of reasonable industry case studies in the production automation domain. Then we designed and implemented a new space-based coordination framework[1] based on the Space Container concept.

4. SCENARIO

In the following, we describe a scenario from the production automation domain (Figure 1). The system consists of several different software agents each being responsible for the machine representing. Such an agent may be

a **pallet agent (PA)** representing the transportation of a production part and knowing the next machine to be reached by the real pallet,

a **crossing agent (CA)** routing pallets towards the right direction according to a routing table,

a **conveyor belt agent (CBA)** transporting pallets, with optionally dynamic speed control, from one crossing agent to another,

a **strategy agent (SA)** which, based on the current usage rate of the production system, knows where to delegate pallets, so that by taking some business requirements, like order situation, into consideration, a product is created in an efficient way,

or a **facility agent (FA)** which specifies the point in time when machines have to be turned off for inspection.

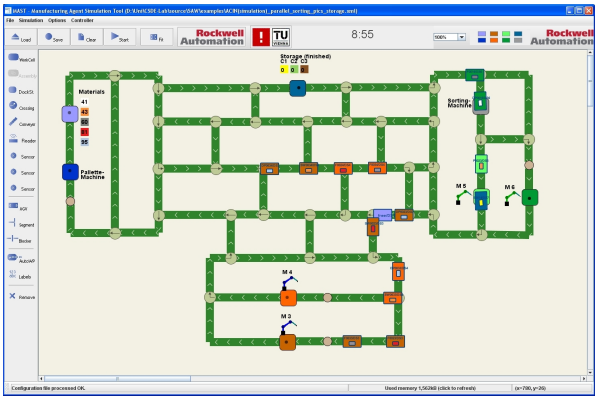

Figure 1: Production Automation System

One of the major challenges in production automation is the need to become more flexible in order to support the fast and efficient reaction to changing business and market needs. However, the overall behavior of the many elements in a production automation system with distributed control can get hard to predict as these heterogeneous elements may interact in complex ways (e.g., timing of redundant fault-tolerant transport system and machines) [17].

[1]to be downloaded at http://www.mozartspaces.org

An approach towards fast reactions may be the prioritization [20, 13] of pallets. Some special parts of the product with higher priority have to be favored by the agents rather than pallets with lower priority. This approach may help to a) produce a small number of products quickly, or b) to phase out products as soon as possible in order to free resources for brand new products to be assembled.

Therefore, the aspect of priority has to be considered between all neighboring CAs and all CBAs connecting them. In the described scenario a CA has to check first, whether there is a pallet with high priority on one of the transporting conveyor belts. If this is the case, that particular CBA may speed up its transportation speed as well as the CA may force the other conveyor belts to stop. This may happen by e.g. either not handling any pallets coming from them and so forcing those CBAs to stop, or by requesting the other CBAs to halt. So, the high priority pallet is being routed earlier than the other pallets, and it has overtaken other pallets which may have occupied machines needed by the prioritized pallet based on its production tree. In section 6 it will be shown how the Space Container concept helps the agents cooperate in order to cope with this requirement.

5. ARCHITECTURE

This section pictures the architecture of a Space Container in detail. It describes the interfaces, supported operations, the way of execution of operations, the handling of Selectors, and the aim of Coordinators.

5.1 Space Container

In its basic form a Space Container is similar to a tuple space. It is a collection of entries accessible via a basic API. The difference is that a Space Container a) may be bounded to a maximum number of entries, b) allows the usage of so called Coordinators (Section 5.3) with each having its specific and optimized view on the stored entries, and c) as in Linda (`out`, `in`, `rd`) it provides an API for reading, taking, and writing entries, but extends the original Linda API with the methods `destroy`, `shift` and `notify`.

5.2 Space Container Architecture

A Space Container is composed of two layers with different functionality. These layer are the Blocking Layer, and the Container Engine. Figure 2 illustrates the architecture of a Space Container and the position of the layers. The Container Engine also has to manage the Coordinators which are as well depicted in the figure as Random-, FIFO-, and PRIO Coordinator. The Random Coordinator contains the references to all existing Entries in the Space Container and returns an arbitrary Entry in case of read operations. The FIFO Coordinator imitates a queue. It stores on the lowest index the reference to the Entry that has been in the Space Container the longest and on the highest index the reference to the Entry that has been added last. The PRIO Coordinator groups references to Entries according to their priority. In the following, each aspect of the figure is described in greater detail.

Space Container Entries: The data items that are stored in a Container are called "“Entries"". An Entry can be either of type Tuple or of type AtomicEntry. A Tuple contains other Entries, which can be either AtomicEntries or other Tuples. An AtomicEntry is a Generic Java class, so when it is instantiated, the class that is contained within

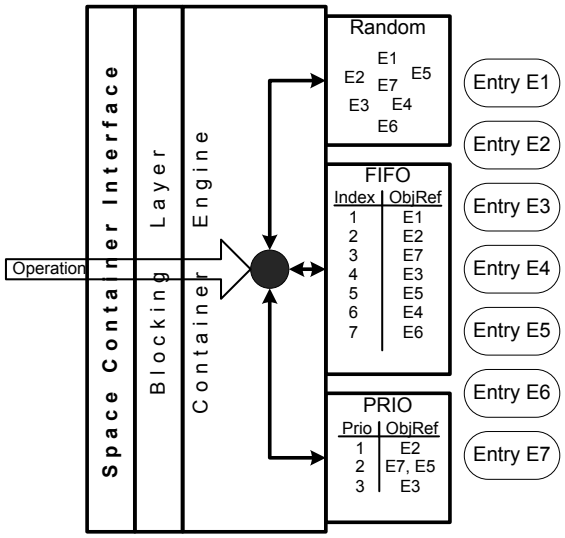

Figure 2: Architecture of a Space Container with a Random-, a FIFO-, and a PRIO Coordinator and 7 entries

the AtomicEntry can be defined.

Space Container Interface: This interface[2] is used by agents to communicate with the Space Container and to execute API operations on the Space Container. Space Containers support bulk operations as well, so that it is possible to insert multiple entries into a Space Container resp. to read/take multiple entries out of it within one operation. The number of entries to be retrieved or to be written is specified in the Selector that is used for the operation.

Beside the fact that read and take operations may be used for synchronization, as both offer a blocking behavior if no suitable entries to be read/taken are in the Space Container, also the write operation may block. A write operation blocks if there is no space left for the Entry in the Space Container or if the same Entry already exists. On the other hand, a shift operation behaves like a write but instead of blocking it will replace as many Entries in the Space Container as many Entries are needed to be written, thus a shift operation is a write operation that never blocks. Which Entries to be deleted, so that the new Entries can be written, depend on the Selectors specified in the operation. In addition, a destroy operation behaves like a take operation but does not return the removed entries.

Every operation that can block has an optional timeout parameter, which represents the maximum time the operation shall be retried in case it has to be blocked. An operation with timeout 0 is executed exactly once and in case it cannot be fulfilled it will not block. An operation with infinite timeout will wait until it can be fulfilled successfully.

Blocking Layer: The blocking layer handles the blocking of operations if this is necessary. The Blocking Layer analyses and executes the operation on the Space Container. If the operation was not successful and has to be blocked, the blocking layer stores the operation in an internal data structure. Later on, when an operation is issued the Blocking Layer removes the operation from its internal data structure and executes it again.

Container Engine: In this layer the management and the correct sequence of execution of the Coordinators is performed. The Space Container Interface supports defining multiple Selectors for one operation. The Container Engine has to make sure that the execution of the operation between the participating Coordinators is done correctly.

5.3 Coordinators

Coordinators are the programmable part of the Space Container and are responsible for managing their view on the Entries in the Space Container. The aim of a Coordinator is to represent a coordination model. Each Coordinator has its own internal data structures which help him to perform its task. Since the coordination model to be realized is known beforehand, the Coordinator can be implemented in an efficient way with respect to its task. Additionally, any number of Coordinators can be added to a Space Container, but a Space Container has at least a Random Coordinator, which is added automatically, to simulate the non-deterministic operation access of Linda.

Coordinators are independent of each other and may belong to one of the following classes:

- Implicit Coordinators have a complete view over all existing Entries in the Space Container. Each Entry that shall be written or removed is passed along with the operation to the Coordinator. Based on only these two pieces of information, the Implicit Coordinator should be capable of managing its view. Coordinators of this class are ones which may want to maintain an order of the Entries in the Space Container, and so representing e.g. FIFO, LIFO, RANDOM, or LRU coordination models.

- Explicit Coordinators are those which require additional meta information for managing the view on the Space Container and their internal data structures. This means that at the time of writing the order of Entries in the Space Container can be influenced from the outside. Furthermore, a certain Entry can be picked out using the given ordering when reading elements from the Space Container. This kind of meta information has to be provided by the user and is passed to the Coordinator by means of the Coordinator's Selector. Examples for such Coordinators may be a Map Coordinator representing a key, value data-structure, where the meta information would be the value of the key, or a Vector Coordinator where the meta information is the position where the Entry should be placed, or Coordinators representing e.g. Binary-Trees, or B-Trees, where the meta information is the index that is used to build up the tree. Other Coordinators may use GPS-Positions as meta information.

- Matching Coordinator are implicit Coordinators with the limitation that they need meta information in case of read operations. The meta information would be in case of a Template Coordinator the template for Linda matching, or an XPath expression in case of an XML Coordinator, or SQL-queries in case a of an SQL Coordinator that is capable of executing such queries on the Space Container.

[2]the complete API JavaDoc can be found at http://www.mozartspaces.org/1.0-alpha/docs/api.html

5.4 Selectors

For every available Coordinator there is a certain Selector that represents the counterpart to this Coordinator. Selectors contain parameters (like a counter for the minimum number of Entries to be retrieved) for queries in case of a read, take, destroy access. In case of writing Entries they contain a) the parameters specifying the appropriate Coordinators and influencing it with special values, and b) the Entry to be added to the Space Container. In case of write operations to Implicit Coordinators there are no additional parameters necessary by definition.

Multiple Selectors can be used within one operation. If more than one Selector is used, the outcome of the first Selector execution will be used as input to the second and so on. Selectors, have a count parameter, indicate the number of entries which have to be returned. For example, if a Random Selector with count 10 is used together with a Key Selector with value "X", the Container Engine asks the Random Coordinator to select ten random entries and afterwards it asks the Key Coordinator to look whether one out of these ten Entries is referenced by the key "X". If an Entry can be found, it is returned. Otherwise, the operation is blocked until the selection can be fulfilled or the timeout of the operation expires.

5.5 Execution of Operations

In the Space Container concept the sequence of Selectors in an operation is AND concatenated. This means that it makes a crucial difference if 10 Entries are randomly selected and then a template matching is performed or that a template matching is done first and then the Random Coordinator tries to find ten Entries non-deterministically. As a consequence, the execution of the operations in the Container Engine is described briefly:

- **write:** first, the write operation is issued on all Explicit Coordinators for which a Selector has been provided. Afterwards, the write operation is executed on all remaining Implicit Coordinators regardless if a Selector has been provided to one of those Coordinators or not.
- **read:** the read operation iterates over the Selectors and invokes the read operation on the corresponding Coordinator. Each Coordinator gets the result of the execution of the previous Selector as an argument. This is necessary because multiple Selectors are evaluated using the Boolean AND operation.
- **take:** this operation uses read operation first to determine which entries have to be deleted. Afterwards, it issues the delete operation on all Coordinators which store a reference on the Entries which shall be deleted.
- **destroy:** is performed exactly as a take operation, but with the exception that the result is dropped in the Blocking Layer.
- **shift:** first, the shift operation is issued on those Explicit Coordinators for which a Selector has been provided. It is possible that an Explicit Coordinator can not decide which Entry to remove. For instance, a Key Coordinator can shift an Entry only if the key which shall be used to store the new Entry is already in use. In this case, the Coordinator throws an exception, and the Container Engine logs that the shift operation could not be performed successfully with this Coordinator. Whether or not, Explicit Coordinators could remove Entries, the shift operation is issued on all Implicit Coordinators. Finally, those Explicit Coordinators are executed again which failed at the first attempt.

6. EVALUATION

In section 3 we have defined two research questions, dealing with the questions whether a) the complexity of coordination can be shifted away from the agent in order to let the agent focus on computational issues and b) the concept of Space Containers allow an efficient implementation and execution of coordination models. In order to answer these questions, we have implemented, based on the scenario from the production automation domain, and evaluated the prioritized queue coordination pattern.

Simplified, the scenario can be summarized as the following: entries have to be ordered by means of the sequence of writing and grouped according to the priority of the entry written. Then, the task is to remove the entry first written from the non-empty group with the highest priority.

Figure 3 depicts on the left side how the Linda space approach would realize the coordination problem. The right side of the figure shows the realization with a Space Container containing a PRIO-FIFO Coordinator. Additionally, both diagrams show the sequence to write an Entry and to take the next Entry with the highest priority from the FIFO perspective.

As it can be seen, the Linda space approach requires much more operations than the Space Container approach. This is because the realization of a prioritized queue requires the modeling of the problem with the resources of Linda and the help of the agent taking over a part of the coordination problem with the following operations in the presented order:

```
Op. 1.: in("in-token", 1, ?int)
Op. 2.: inp("msg", 1, 3, ?P )
Op. 3.: out("in-token", 1, 3)
Op. 4.: in("in-token", 2, ?int)
Op. 5.: inp("msg", 2, 3, ?P )
Op. 6.: out("in-token", 2, 4)
```

In order to get the Entry with the highest available priority, the tuple space approach must traverse the priorities, starting with the highest. To do so, each priority needs a special token, so that concurrent agents can access the space in a synchronized way. The token for priority one is taken (Op. 1) retrieving the index (3) of the first Entry in the queue. Next, a pattern matching for messages (ḿsg)́ with priority 1 and index 3 is initialized (Op. 2). Since the attempt to take was not successful the process is started again from the beginning, but in this iteration the next higher priority is used. If an entry was found (Op. 5) the index of the queue is increased by one and written back into the space.

In case an Entry should be added to the queue, the following steps have to be performed by an agent:

```
Op. 7.: in("out-token", 2, ?int)
Op. 8.: out("msg", 2, 5, P )
Op. 9.: out("out-token", 2, 6)
```

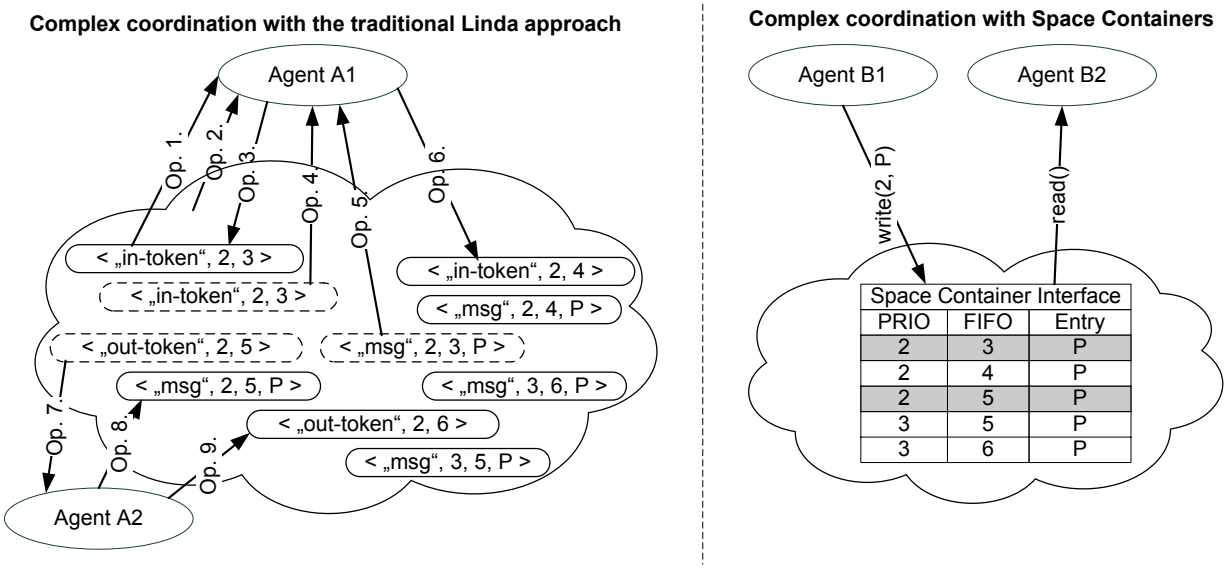

Figure 3: Comparing the complexity of a prioritized queue of the traditional Linda space approach with the Space Container concept (P..payload)

The last index of the queue has to be retrieved (Op. 7), the message with the appropriate priority and index written (Op. 8), and the retrieved token, with increased index by one, written back.

On the other hand, the Space Concept requires the execution of one operation only. In case of retrieving the next Entry with the highest priority, the agent has to know the name of the Coordinator, and execute the operation (Op. S2).

```
Op. S1.: write(new PRIOFIFOSelector(2,P))
Op. S2.: read(new PRIOFIFOSelector())
```

In case of writing, the agent has to know the name of the Coordinator, and the priority of the payload. As it can be seen, the complexity of the agent, the number of operations to be executed, could be reduced by shifting the complexity of coordination into the Space Container.

The second part of the evaluation deals with the question if the concept of Space Containers allow an efficient implementation and execution of coordination models due to the distinction between the data needed for coordination purposes only and the payload.

Entries	Linda	Improved-Linda	PRIO-FIFO
10000	5.24ms	0.41ms	0.20ms
20000	15.15ms	0.50ms	0.20ms
30000	47.93ms	0.57ms	0.21ms
40000	58.66ms	0.63ms	0.20ms
50000	70.10ms	0.66ms	0.21ms

Table 1: Comparing the time of retrieving a single Entry using the prio-queue coordination principle implemented by means of different Space Container Coordinators

Therefore, we measured the time that is required to retrieve the next Entry, with highest priority, from a prioritized queue. A benchmark has been set up, which compares the performance of a simple Linda Coordinator, a Linda Coordinator with improved data structures and a PRIO-FIFO Coordinator. The benchmark should demonstrate that by using the knowledge about the aim of prioritized queues a PRIO-FIFO Coordinator is able to retrieve Entries faster than a Coordinator with Linda pattern matching techniques. The difference between the Linda Coordinator and the improved-Linda Coordinator is the adapted data structure used to store Entries. The Linda Coordinator is using a common HashMap, with the entries' ID as key, whereas the improved-Linda Coordinator uses a data-structure optimized to the characteristics of Linda template matching taking the signature of the tuple into account.

In order to run the benchmark the Space Container was first filled with a specific amount of Entries (10000, 20000, 30000, 40000 and 50000 Entries). After that a single read operation was issued, and the time needed to get the Entry measured. The machines characteristics on which the benchmarks were executed are listed in table 2.

Processor	AMD Athlon™ 64 X2 Dual Core 6400+ (each core with a frequency of 3.2GHz)
Main memory	2GB
OS	Ubuntu 8.04 32bit
Java™ Version	1.6.0_07-b06

Table 2: Machine's characteristics used for all benchmarks

The results of the benchmark are presented in table 1, clearly showing that the PRIO-FIFO Coordinator is constantly the fastest of all three, followed by the improved Linda Coordinator. Due to the usage of a hashtable in the Linda Coordinator the results of the benchmark are not linearly increasing. The reason is that the hashvalue is computed from the Entry's ID, which is changing each time an Entry is instantiated, and which specifies the position of the

Entry in the hashtable.

7. CONCLUSION AND FUTURE WORK

In this paper, we described the concept of structured spaces, so called Space Containers, which allow to store tuples in a customizable structured way. We derived two research questions and answered them based on the evaluation of a scenario from the production automation domain:

Concept of Space Containers: The concept of Space Containers has moved the complexity in case of coordination models with ordering characteristics away from the agent to the proposed coordination framework. The complexity of a coordination issue is concentrated at one point enabling a clear separation between the computation model and the coordination model again. By moving the complexity into the Space Container the exemplary scenario can be reduced to a single operation call.

Efficient implementation of coordination patterns. The evaluation showed that the distinction between coordination data and payload improves the efficiency of coordination significantly. This is due to the fact, that a) the aims of the Space Container is known beforehand and therefore it can be implemented efficiently with respect to the coordination model it is representing, and b) the coordination data can be used additionally to optimize data retrieval from the Space Container.

Future work contains the realization of more sophisticated coordination patterns [10, 6], like the auction or market place pattern, based on the proposed Space Container approach.

8. REFERENCES

[1] G. Cabri, L. Leonardi, and F. Zambonelli. Reactive tuple spaces for mobile agent coordination. In *MA '98: Proceedings of the Second International Workshop on Mobile Agents*, pages 237–248, London, UK, 1998. Springer-Verlag.

[2] G. Cabri, L. Leonardi, and F. Zambonelli. Mars: a programmable coordination architecture for mobile agents. *Internet Computing, IEEE*, 4(4):26–35, Jul/Aug 2000.

[3] S. Capizzi. A tuple space implementation for large-scale infrastructures. Technical report, Department of Computer Science, University of Bologna, March 2008.

[4] N. Carriero and D. Gelernter. Linda in context. *Commun. ACM*, 32(4):444–458, 1989.

[5] M. Cremonini, A. Omicini, and F. Zambonelli. Coordination and access control in open distributed agent systems: The tucson approach, 2000.

[6] D. Deugo, M. Weiss, and E. Kendall. Reusable patterns for agent coordination. pages 347–368, 2001.

[7] E. Freeman, K. Arnold, and S. Hupfer. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley Longman Ltd., Essex, UK, UK, 1999.

[8] D. Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[9] D. Gelernter and N. Carriero. Coordination languages and their significance. *Commun. ACM*, 35(2):97–107, 1992.

[10] S. C. Hayden, C. Carrick, and Q. Yang. A catalog of agent coordination patterns. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 412–413, New York, NY, USA, 1999. ACM.

[11] N. R. Jennings and M. J. Wooldridge. *Agent Technology: Foundations, Applications and Markets*. Springer Verlag, 1998.

[12] Y. Jiang, G. Xue, Z. Jia, and J. You. Dtuples: A distributed hash table based tuple space service for distributed coordination. *Grid and Cooperative Computing, 2006. GCC 2006. Fifth International Conference*, pages 101–106, Oct. 2006.

[13] K. Kemppainen. *Priority scheduling revisited - dominant rules, open protocols and integrated order management*. PhD thesis, Acta Universitatis oeconomicae Helsingiensis. A, December 2005.

[14] M. Kinga and C. Adrian. Glinda - grid-based distributed linda system. *Symbolic and Numeric Algorithms for Scientific Computing, 2007. SYNASC. International Symposium on*, pages 349–352, Sept. 2007.

[15] R. Kishore, H. Zhang, and R. Ramesh. Enterprise integration using the agent paradigm: foundations of multi-agent-based integrative business information systems. *Decision Support Systems*, 42(1):48–78, Oct. 2006.

[16] E. Kühn, R. Mordinyi, and C. Schreiber. An extensible space-based coordination approach for modeling complex patterns in large systems. *3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, Special Track on Formal Methods for Analysing and Verifying Very Large Systems*, 2008.

[17] A. LÄijder, J. Peschke, T. Sauter, S. Deter, and D. Diep. Distributed intelligence for plant automation based on multi-agent systems: the pabadis approach. *Production Planning and Control*, 15:201–212, 2004.

[18] T. J. Lehman, A. Cozzi, Y. Xiong, J. Gottschalk, V. Vasudevan, S. Landis, P. Davis, B. Khavar, and P. Bowman. Hitting the distributed computing sweet spot with tspaces. *Comput. Netw.*, 35(4):457–472, 2001.

[19] A. L. Murphy, G. P. Picco, and G.-C. Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Trans. Softw. Eng. Methodol.*, 15(3):279–328, 2006.

[20] C. Rajendran and O. Holthaus. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research*, 116(1):156–170, July 1999.

[21] R. Tolksdorf, F. Liebsch, and D. M. Nguyen. Xmlspaces.net: An extensible tuplespace as xml middleware. In *In Report B 03-08, Free University Berlin, ftp://ftp.inf.fu-berlin.de/pub/reports/tr-b-0308.pdf, 2003. Open Research Questions in SOA 5-25 and Loose Coupling in Service Oriented Architectures*, 2004.

[22] R. van der Goot, J. Schaeffer, and G. V. Wilson. Safer tuple spaces. In *COORDINATION '97: Proceedings of the Second International Conference on Coordination Languages and Models*, pages 289–301, London, UK, 1997. Springer-Verlag.

[23] G. Wells, A. Chalmers, and P. Clayton. Extending the matching facilities of linda. In *COORDINATION '02: Proceedings of the 5th International Conference on Coordination Models and Languages*, pages 417–432, London, UK, 2002. Springer-Verlag.

[24] G. C. Wells. Coordination languages: Back to the future with linda. *Proceedings of the Second International Workshop on Coordination and Adaption Techniques for Software Entities (WCAT05)*, pages 87–98, 2005.

[25] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. T spaces. *IBM Systems Journal*, 37(3):454–474, 1998.

[26] D. Xu, X. Bai, and G. Dai. A tuple-space-based coordination architecture for test agents in the mast framework. In *SOSE '06: Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering*, pages 57–66, Washington, DC, USA, 2006. IEEE Computer Society.