Bridging the Gap with P2P Patterns

Alois Ferscha[1], Manfred Hechinger[1], Rene Mayrhofer[1]
Ekaterina Chtcherbina[2], Marquart Franz[2], Marcos dos Santos Rocha[2], and Andreas Zeidler[2]

[1] Insitut für Pervasive Computing, Johannes Kepler Universität Linz, Altenberger Straße 69, A-4040 Linz, Austria {ferscha,manfred,rene}@soft.uni-linz.ac.at
[2] Corporate Technology CT SE 2, Siemens AG, Otto-Hahn-Ring 6, 81739 Munich, Germany {marcos.rocha, a.zeidler, ekaterina.chtcherbina, marquart.franz}@siemens.com

Abstract The design principles of pervasive computing software architectures are widely driven by the need for opportunistic interaction among distributed, mobile and heterogeneous entities in the absence of global knowledge and naming conventions. Peer-to-Peer (P2P) frameworks have evolved, abstracting the access to shared, while distributed information. To bridge the architectural gap between P2P applications and P2P frameworks we propose patterns as an organizational schema for P2P based software systems. Our Peer-it hardware platform is used to demonstrate an application in the domain of flexible manufacturing systems.

Keywords: Peer-to-peer computing, pervasive computing, application patterns, Mobile computing, Ad-hoc interaction

1 Introduction

Peer-to-Peer systems have recently gained more and more interest for various applications scenarios. Unlike in the area of object oriented software design, patterns for the development of peer-to-peer applications are not generally known. The motivation for developing patterns for peer-to-peer systems is twofold: First, P2P systems, and especially ad-hoc peer-to peer systems, are complex due to mobility of the interconnected nodes, the lack of dedicated master nodes and in general due to the involvement of a potentially large number of nodes. The mobility of peers in space, connectivity and mode of operation creates a highly dynamic system with manifold issues like transient connectivity, network splits with disjoint groups, resending of messages, dynamic loop detection, transient life- and dead-locks, etc. Other reasons for the complexity are a large number of nodes that should be coordinated and the lack of dedicated master or synchronization nodes in P2P systems. Issues arising from these complexities have been studied extensively in the area of coordination models and languages. Secondly, developing real-world P2P applications is laborious due to the combination of often heterogeneous peers and the need for P2P concepts to span multiple layers of abstraction (cf. ISO/OSI model). Many current P2P applications are developed from scratch and are thus concerned with basic communication layers as well as the application logic itself.

P2P patterns can address both factors by, on the one hand, assisting the application designer to cope with the inherent complexity and therefore decreasing the probability of errors, and, on the other hand, providing ready-to-use parts of P2P systems so that application designers can concentrate on the application logic and are relieved of the issues in lower layers. To provide a pool of typical P2P applications flows in mobile ad-hoc networks using an appropriate P2P system for discovery and communication, we present several P2P application patterns within this document. These application patterns are situated between P2P infrastructures / frameworks and applications themselves, as depicted in Figure 1. Typically, P2P infrastructures implement the discovery of peers as

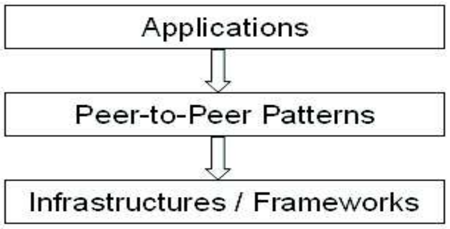


Figure 1. Scope of P2P Patterns

well as resources (for example music files) and the communication with peers as there core functionality. Numerous P2P infrastructures and frameworks have already been developed, therefore P2P applications can already be built on top of various existing infrastructures, whichever is situated best for the corresponding application. Examples for such frameworks are JXTA [1], NKF [2], or more static approaches like Freenet [3]. Applications which are built on top of P2P infrastructures typically contain the handling of sensors and actuators, user interfaces and application logic. Various applications are currently available, ranging from file sharing (e.g. Gnutella [4] or BitTorrent [5]) and folder synchronization (e.g. Novell iFolder [6]) to voice-over-IP systems (e.g. Skype [7]) or gaming platforms (e.g. DirectPlay) and many more.

For our work, we use the P2P Coordination Framework [8] as the basis. It provides a comprehensive infrastructure for P2P applications in fully decentralized mobile ad-hoc scenarios and implements discovery of peers, communication between peers independently of the used communication technology, limiting communication to spatial proximity using a spatial proximity sensor, transparent interaction with resource-less objects [9], transparent invocation of remote services [10] and distributed coordination mechanism utilizing a context sensitive profile description language [11]. Therefore, the framework is a basis for P2P applications in decentralized and mobile ad-hoc scenarios; it does however not directly implement application logic for such domains. For the remainder of this paper, we define mobile ad-hoc P2P systems as a set of partially interconnected nodes which shall be coordinated to solve a given problem. These nodes are mobile in space, connectivity and mode of operation and interact without prior knowledge of each other upon contact defined by the communication media. P2P patterns as introduced in this document solve recurring coordination problems of mobile ad-hoc P2P systems.

2 Related Work

The definition of patterns, as used in software development today, origins from Alexander et al. [12] and describes patterns as follows: *"Each pattern describes a problem with occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice".*

In [13], Gamma et al. explain the important distinction between frameworks and patterns such that patterns are 1) more abstract 2) smaller architectural elements and 3) less specialized than frameworks. The separation towards algorithms is provided by [14] such that algorithms are more fine-grained than patterns. [15] categorizes patterns by its abstraction level into "architectural patterns", "design patterns" and "idioms", where architectural patterns "*... express a fundamental organization schema for software systems ...*". Another important category are "interaction patterns" as, for instance, presented by IBMs patterns for e-business application interaction [16]. Similar to these, we also address application-level patterns for rapidly building P2P applications.

The most similar work to P2P patterns for Applications can be found in the JXTA services layer, which is built on top of the JXTA core layer and according to [1], deals with "*higher-level" concepts such as indexing, searching, and file-sharing. [...] These services [...] are useful by themselves but are also commonly included as components in an overall P2P system.* These services are used to build JXTA applications, similar to the patterns proposed within this paper. The general idea and concept of several projects in the JXTA service layer might also be good candidates for patterns. Examples projects are "dO", that implements distributed (replicated) objects using JXTA, "jxtaspaces", which aims to provide a distributed shared memory service for JXTA, "meteor", implementing a P2P distributed hash-table, "search", which is a distributed search service on top of JXTA and "jngi", which is a Framework for job distribution among several peers. These services are however specific implementations for Project JXTA and can not be immediately applied for other P2P infrastructures.

3 P2P Patterns

Within this Chapter, we present several example patterns. We intentionally tried to model P2P patterns resembling well-known OO patterns for ease-of-use.

3.1 The Checklisting Pattern

The Checklisting pattern provides means for maintaining a (usually ordered) list of services (usually running on different peers) an application has to interact with, considering arbitrary constraints. This problem typically arises in mobile scenarios where a task is accomplished by the collaboration of multiple peers.

To solve the tasks sketched above, the Checklisting pattern proposes the following entities, which are illustrated in Figure 2.

– Check-Items are the individual "items" that must be "checked". Each Check-Item has an associated action, which has to be accomplished by the Checker-Peer in order to get the Check-Item checked.
– Checklists contain a (usually ordered) list of Check-Items and can be interpreted as a project network diagram. The individual items are connected via edges which express a causal dependency between Check-Items (for example that a Check-Item A must be accomplished before another Check-Item B). Each Check-Item additionally may have arbitrary associated constraints which are considered while processing the Checklist.
– The Checklist-Service is responsible for managing Checklists and for the interaction with other peers (in particular with Checker-Peers) in order to get the Check-Items of a Checklist accomplished.
– Checker-Peers provide means for performing arbitrary actions (for example a processing step). They are used to tick off Check-Items after the corresponding actions have been executed.

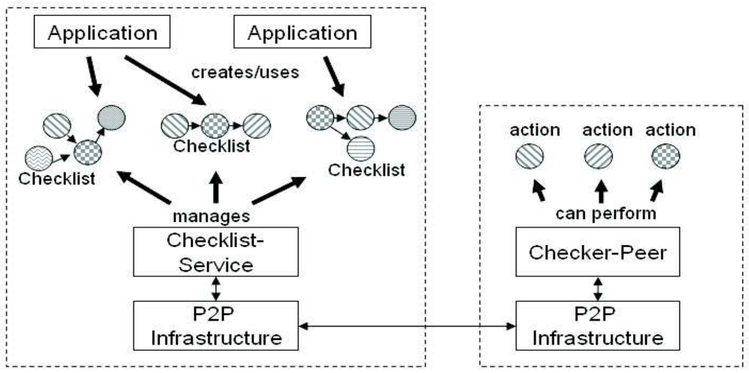


Figure 2. Checklisting pattern entities

For modeling the interaction requirements of an entity, Checklists are used. Checklists are similar to techniques for network analysis. The actual features of the Checklist regarding the interconnection of nodes (for example alternatives) and the use of constraints (various constraint types) may vary. Each node,

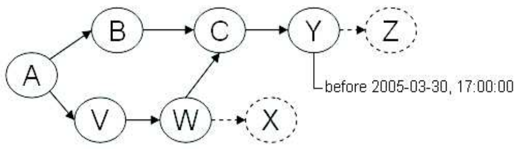


Figure 3. An example checklist

as well as each edge, may have several constraints assigned (cf. Figure 3). A violated constraint is reported automatically by the Checklist-Service in order to allow the application to incorporate according handling.

An application typically gets the items of a Checklist accomplished by executing the following operations:

1. Create a Checklist by subsequently adding Check-Items.
2. Get the first/next Check-Item that must be checked
3. Wait until a Checker-Peer capable of checking the item is available
4. Start checking the item, which usually means that the action of the Check-Item is executed on the Checker-Peer.
5. If any further unchecked Check-Item is available, go to step two.

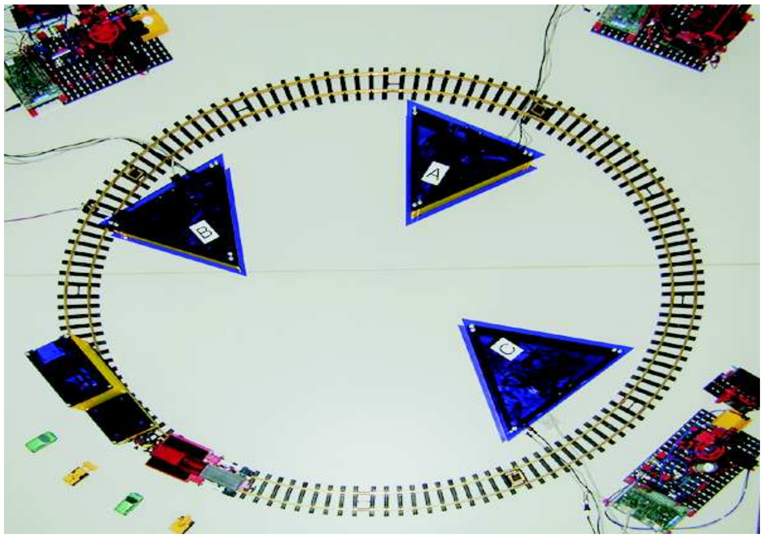
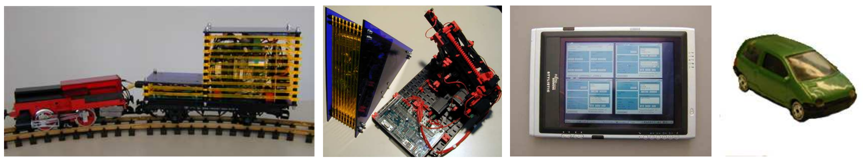


Figure 4. Example checklisting scenario



(a) Transport peer (b) Processing peer (c) Monitoring peer (d) Artifact

Figure 5. Basic building blocks of the flexible manufacturing scenario

Example Scenario The example scenario illustrated in Figure 4 heavily relies on the Checklisting pattern. It implements a flexible manufacturing system (*FMS*), where the processed items are aware about themselves and autonomously determine the next processing machine they have to visit. The depicted flexible manufacturing cell consists of multiple processing machines (A, B, C), a transport vehicle and different processing goods, which we call artifacts. Figure 5 shows these four basic building blocks in their current implementation for our research lab: the transport peer is an autonomous, self-moving vehicle that carries artifacts from/to processing peers; the processing peers are robots that offer different services to the FMS and execute processing steps on artifacts; the monitoring peers is an arbitrary tablet PC or notebook and serves as a GUI for the operator to monitor and control the whole FMS; artifacts are the workpieces which are moved by the transport peer and manipulated by the processing peers, i.e. which are handled by the manufacturing process.

Each artifact maintains a checklist containing various processing steps that must be accomplished by "visiting" different processing machines. Artifacts utilize the depicted transport system to get transported to those processing machines providing the processing service(s) stored in the checklist. After all check-items of the checklist are successfully checked, the processing good is transported to its final destination.

Figure 7 depicts a flow of an artifact in a manufacturing cell. The artifact maintains the checklist depicted in Figure 6. Processing is started on machine

$$\begin{tikzcd}
 & B \arrow[dr] & & & \\
A \arrow[ur] \arrow[dr] & & A \arrow[r] & C \arrow[r] & B \\
 & C \arrow[ur] & & &
\end{tikzcd}$$

Figure 6. Scenario checklist

A, then requires machine B and C (or C and then B). If both items (B and C) are checked, the processing machines A, then C and finally again B are required.

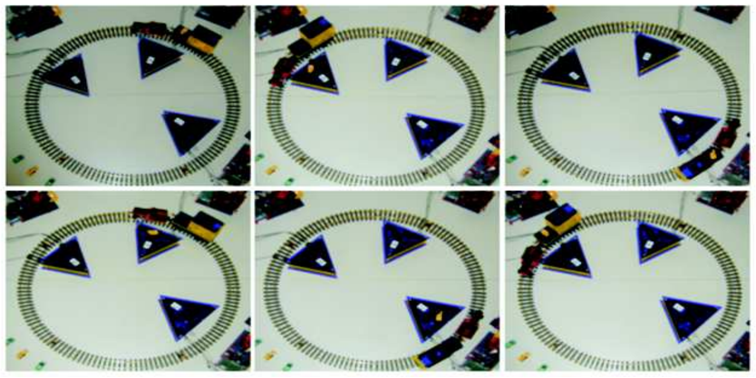


Figure 7. Checklist Processing in a FMS Scenario

3.2 A General P2P Pattern Prospect

Functional and Topological patterns We have identified two main categories of P2P patterns, “Functional” and “Topological” patterns. Functional patterns directly assist applications accomplishing their respective tasks by providing implementations for specific recurring problems regarding the interaction among two or more peers. Functional patterns usually require that they are implemented on each affected peer. The Checklisting Pattern proposed in the previous chapter is a typical functional pattern.

Topological patterns in contrast, usually affect the “network topology” of the involved peers or adapt the architecture of the distributed system consisting out of several peers. Topological patterns for example create virtual nodes, aggregate nodes to “supernodes”, split physical nodes to multiple virtual nodes or create partitions of nodes.

The rest of this chapter briefly motivates other possible pattern in the area of P2P architectures.

Ad-hoc P2P Introspection An Ad-hoc Introspection Pattern could provide means for the recurring problem of introspecting data on remote peers in mobile ad-hoc networks. Such a pattern could ease the query of approved status variables of a peer. A manfucating scenario as sketched in the previous chapter could include maintenance workers which survey the progress and status of the manufacturing cell. The Ad-hoc P2P Introspection pattern is a typical functional pattern.

Intermediary Peer Connectivity and reachability is an important topic in P2P sytems. If two peers use different communication technologies (e.g. Bluetooth, WLAN) or a peer is not reachable due to its limited communication range, an Intermediary Peer, providing transparent access to resources of one ore more other peers, could provide means for improved connectivity. The Intermediary Peer pattern is a topological pattern.

P2P Synchronization The P2P Synchronization pattern is a functional P2P pattern which can be used for the synchronization of arbitrary data between multiple peers. It provides a solution sketch for the problem of having a consistent view of multiple data objects in P2P environments.

P2P Chain of Responsability This functional P2P pattern candiate represents a solution for the delivery of "events" in P2P networks without the need for specific device addressing, similar to content-based messaging algorithms. We propose to use event descriptions and interest descriptions to deliver events to those peer(s) with the most matching interest description. The pattern could be used for sensor networks to provide a base for event notification without the need for preliminary subscription at "sensor peers".

Visitor We propose the Visitor pattern for the transfer and execution of arbitrary active code in P2P ad-hoc networks. In such a pattern, an active component could decide on its own to which peer it whishes to "migrate". Since executing unknown active code raises serious security problems, this pattern should only provide restricted access to the resources of the peer. We propose that the process of migration from one peer to another includes the following tasks: 1) Receiving the active component 2) Installing the component and preparing it for execution 3) Starting of the component within a secure environment 4) Packing the component and results to a transferable unit and finally 5) Transferring the unit to another peer as requested by the active component.

4 Conclusion

Software architectures for the spontaneous interaction and interoperation of mobile, ad-hoc, context aware and autonomous entities are increasingly based on low level of abstraction P2P frameworks, thus challenging the P2P application development process. In this work we have motivated why it is important to address the architectural gap among P2P applications and the underlying P2P frameworks and infrastructures in a systematic and structured way. We have proposed a design patterns based approach, and have discussed and developed P2P design patterns. Besides considerably easing the development process, the pattern oriented approach gains from the reduced efforts for solving coordination problems and improved application stability due to the reuse of proven P2P code.

References

1. Gong, L.: Project JXTA: A technology overview. Technical report, Sun Inc. (2001)
2. Roth, J.: A communication middleware for mobile and ad-hoc scenarios. In: Proceedings of the International Conference on Internet Computing (IC'02). (2002) 77–84
3. Clarke, I., Miller, S., Hong, T., Sandberg, O., B.Wiley: Protecting free expression online with Freenet. IEEE Internet Computing **6** (2002) 40–49
4. Bordignon, F., Tolosa, G.: Gnutella: Distributed system for information storage abd searching, model description. IT-Journal of Internet Technology **2** (2001)
5. Cohen, B.: Incentives to build robustness in BitTorrent (2003)
6. Novell: Novell iFolder. Technical white paper, Novell Inc. (2002)
7. Skype: The Skype internet telephony homepage (2004) http://www.skype.org.
8. Ferscha, A., Hechinger, M., Mayrhofer, R.: The peer-to-peer coordination framework - architecture reference. Technical report, Institut für Pervasive Computing, Johannes Kepler Universität Linz (2004)
9. Mayrhofer, R., Ortner, F., Ferscha, A., Hechinger, M.: Securing passive objects in mobile ad-hoc peer-to-peer networks. In Focardi, R., Zavattaro, G., eds.: Electronic Notes in Theoretical Computer Science. Volume 85.3., Elsevier Science (2003)
10. Ferscha, A., Hechinger, M., Mayrhofer, R., Oberhauser, R.: A light-weight component model for peer-to-peer applications. In: Proceedings of the 2nd International Workshop on Mobile Distributed Computing (MDC04), IEEE Computer Society Press (2004) 520–527
11. Armbruckner, M.: Multi-User Interaktion in ad-hoc Netzen. Master's thesis, Johannes Kepler Univseritaet Linz, Austria (2002)
12. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl, I., Angel, S.: A Pattern Language. Towns,Buildings,Construction. Oxford University Press (1977)
13. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley Publishing (1995)
14. Appleton, B.: Patterns and software: Essential concepts and terminology (2000) http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html.
15. Schmidt, D., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects. Volume 2. John Wiley and Sons (2000)
16. IBM: IBM patterns for e-business (2004) http://www-106.ibm.com/developerworks/patterns/index.html.