A Tuple-Space based Middleware for Collaborative Tangible User Interfaces

Florian Georg Furtmüller  
*University of Linz*  
florian.furtmueller@gmx.at

Stefan Oppl  
*Department of Business Information Systems*  
*Communications Engineering*  
*University of Linz*  
stefan.oppl@jku.at

**Abstract**

Several approaches have been made to establish a generic middleware for tangible user interfaces (TUI). They target toward the independence of application domains as well as flexibility in terms of the deployed sensor technology. For collaboration support TUIs have to allow the spatial distribution, asynchronous activities, and the dynamic modification of the TUI infrastructure, to name the most prominent ones. In this paper we present a framework approach based on the LINDA tuple space concept to meet these requirements. The implemented TUIpist framework deploys arbitrary sensor technology for any type of application and actuators in distributed environments. We demonstrate a use case, in order to proof the concept in actual work settings, and to demonstrate the adaptability of the approach.

1. Introduction

Tangible User Interfaces (TUIs) support cooperation among individuals (e.g. [14], [17]). The rapid development of embedded, low-cost sensor devices and wireless communication technology facilitates access to a variety of information, which can be used to interpret users’ interactions with the interactive systems.

However, the infrastructure development for TUI applications always required to get “down and dirty” [17] with sensor input technology and low level software design. The high requirements on technical expertise prevent broad deployment of TUI applications, as it was the case for GUI development 20 years ago, before the first GUI frameworks were presented (ibid.).

Researchers becoming aware of that deficiency (cf. [7][8][12][17][19]) have built middleware to abstract from input technology and supply applications with information on the user’s actions in a technology-independent format. The most common approach is event-oriented coupling of data-input and applications, as known from GUI frameworks: a user's action (e.g. a keystroke) triggers an event, on which applications can react. Events are independent of the actual technology, which senses the user's action (e.g. for keystrokes: the keyboard or a touchscreen).

Middleware following this approach has been successfully deployed in collaboration support settings (e.g. [8][12][17]) with static IT-infrastructure, where producer and consumer of information are known in advance.

However, supporting collaboration in all its aspects requires considering co-located and spatial distributed settings as well as synchronous and asynchronous collaboration [13], where participants may also change over time. Technically speaking, this leads to an environment, where (a) data can be provided or consumed by an arbitrary number of instances, (b) the instance may join and leave over time and (c) possibly have different requirements on kind and amount of provided information. Event-based approaches in this context have some shortcomings [25], due to the required flow of control for dynamic reconfiguration.

In this paper, we try to meet these challenges by proposing a data-centric approach to TUI-state representation. The basic building blocks of TUI middleware as identified in related work (see section 2) are used for data-driven coordination to meet the requirements given above. We have implemented this approach establishing the TUIpist-framework [9].

Section 3 presents the concepts that were applied when designing the coordination framework. In section 4, implementation of the framework and application during TUI design time and in operation are described. Section 5 shows how the framework has been used in practice. Finally, we conclude with a summary of achieved results and some inspirations for future work.
2. Related Work

For the development of the TUIpist-framework, we have reviewed earlier approaches in terms of concepts and implementation strategies. The projects, which have influenced our approach, are described in the following:

Context Toolkit

The Context Toolkit project [7] aimed at providing means for more structured implementation of contextual applications. The definition of generic building blocks required for context information processing and the introduction of different levels of data abstraction (from raw to interpreted and aggregated context information) have inspired the basic components of the TUIpist-framework. In contrast to the TUIpist-framework, which is based on Jini (thus following a distributed approach), the CTK pursues a server-centric approach, which introduces a single point of failure. Further, every sensor needs to be administrated individually by a widget, which may lead to scalability problems [2].

Papier-Mache

Papier-Mâché [17] has been considerably inspired by the Context Toolkit. It is one of the first implementations of a toolkit for tangible interfaces. The main objective was to allow rapid prototyping of tangible interfaces and transparent exchange of sensor technology (e.g. for using visual sensors during design time and switching to RFID sensors for deployment).

TUIpist builds upon the Phob (Physical Object)-concept introduced by Papier-Mâché. Phobs are the basic unit of data representation and directly map physical artifacts to virtual representations. However, Papier-Mâché still follows an event-driven approach to coordination among components. In TUIpist, Phobs are used as the central unit of coordination among the component.

MUSE

MUSE is a middleware-architecture to support the development of contextual real-time applications and services for sensor-equipped smart environments [5]. MUSE, like the TUIpist-Framework, uses Jini infrastructure technology as service platform. The sensor service in MUSE is organized similarly to the sensor components of TUIpist; the fusion service provides similar functionality like the ApplicationMediator in the TUIpist-framework. The relation between Sensor and Sensor Services is similar organized like the CTK and is realized with a 1:1 cardinality. This will lead to problems in scalability.

MUSE does not either offer an opportunity for persistent data storage e.g. for later evaluation [19].

3. Concept

Motivated by earlier approaches for ubiquitous interaction infrastructures [18] and orchestration of mobile pervasive computing platforms [20], we have chosen the LINDA tuple-space-concept [4] as a foundation for our coordination framework.

LINDA is a means to coordinate processes solely via data contained in a ‘tuple space’ [6]. A tuple space can be interpreted as a virtual blackboard, which is used by several processes for communication and/or interaction. The participating processes do not communicate via messages and do not share variables. Rather, they produce data structures, called ‘tuples’, and place them into the tuple space for access by other processes [3]. This access is realized using three basic operations:

- out(t) Create a new tuple and put it into the tuple space.
- rd(t) Read a tuple from the tuple space without removing it.
- in(t) Read and remove a tuple from the tuple space.

Linda thus can be understood as a producer-consumer algorithm, which is solely orchestrated using data structures [3].

Consequently, when designing our framework, we took a data-centric perspective. Coordination between input devices, output devices and orchestration among them is founded solely on the data collected and processed. Abstraction of physical object properties (like position, orientation, color, weight, etc.) to software-processable values is realized (domain-specifically) by interpretation and combination of the data output of various input devices.

Figure 1 gives an overview of the components of the TUIpist-Framework. The basic customizable building blocks are Sensors, Aggregators and Applications. The concepts behind those components are pretty similar to those given in the Context Toolkit [7], but differ in details:

**Sensors**

Sensor devices supply the framework with information about physical objects (e.g. position, rotation...). Sensor components are used to attach sensors to the framework. Raw sensor data is preprocessed and interpreted here. Sensor components generate tuples representing physical artifacts (cf.
which contain information collected by this sensor about the artifact.

Aggregators

Aggregators are used to collect, combine and interpret the information about physical artifacts delivered by the different sensors. Aggregators consequently have to be designed specifically for every type of artifacts to be handled. They read sensor-generated tuples from the space and put back aggregated tuples containing all information about the state of a physical artifact.

Applications

All parts of the TUI, which have to be supplied with data, are considered applications (including possible actuators used in the TUI). Components connecting an application to the framework read aggregated tuples from the space and possibly filter and recode information according to the application's requirements.

Figure 2 shows how the framework orchestrates the components using the tuple space. The figure depicts a setting with two types of physical artifacts, which are sensed by three sensors and are used by one application. The grayed areas in the tuple representations show the aspects already known about the artifact. As can be seen, Sensor 1 provides the majority of information about the artifacts, which are represented with hexagonal shapes here. Aggregator 1 takes this information together with that provided by Sensor 2 to produce a tuple, which contains all relevant aspects of information of the represented artifact. Sensor 3 does not contribute to this type of artifacts. The Application only takes already aggregated information for further processing.

3. Implementation & Application

As defined in the requirements, the TUIpist-framework has been designed to work in distributed and non-distributed environments. Jini [28] has been used to provide the basic service infrastructure. The Java-based middleware provides simple and transparent management of distributed framework environments. Additionally, Jini has already been used for some implementations of the LINDA concept.

JavaSpaces

In order to provide independence from concrete implementations of the tuple space concept, the TUIpist-framework follows the principle of using only standard tuple space operations (out, read, in – as described above). This to change the implementation of the tuple spaces, as long as it is compliant to the JavaSpaces reference implementation [27]. At present, we use the tuple space implementation of GigaSpaces [11] for reasons of scalability and efficiency. Providers of other tuple space implementations are e.g. IBM TSpaces [15] or the open source project Blitz [1] (both not based on Jini).

Application of Framework

Figure 3 shows the internal architecture of the framework. Sensors are connected to the framework using an adapter – the SensorFacade – which delivers raw data to a PhysicalObjectFactory. This factory interprets the raw data and generates or updates a PhysicalObject-tuple in the space or removes it from the space, respectively.

Aggregator components have to be implemented for every type of physical artifact based on the InformationAggregator extension point. Whenever an InformationAggregator encounters a new or updated
Use of Design Patterns

Object-oriented design patterns [10] are commonly accepted means to construct highly structured software that is easy to grasp. Hence, TUIpist has been developed using design patterns. In this way, the framework can easily be extended, and its components can easily be modified and exchanged with others.

As an example for the concrete use of design patterns in the TUIpist-Framework, Figure 4 depicts the pattern for sensor interface and two implementing classes (for a camera and magnetic field sensor).

If it is necessary to add a new sensor to the framework, this is realized by simply implementing a new façade class derived from the abstract class sensor façade.

Additionaly, the factory pattern has been used for the PhysicalObjectFactory, the mediator pattern has been used for the ApplicationMediator and the strategy pattern has been used to build the InformationAggregator.

Figure 3. Internal Framework Structure

PhysicalObject of the artifact type they handle, it either creates a new AggregatedObjects or merges the new information with that provided previously by other sensors. The AggregatedObjects are then put in the tuple space or are updated there.

Applications have to be registered to the framework using an ApplicationMediator component. This component looks for required information about physical artifacts, filters and possibly recodes it and provides it to the application using an ApplicationFacade.

For reasons of sound conceptual framework design, these components have been marked ‘Hot Spots’ or ‘Frozen Spots’ [26]. Hot Spots describe extension points whereas Frozen Spots mark the not modifiable parts of a framework. The definition of these spots as well as the use of object oriented design patterns [10] facilitates an easier application of the framework [26].

All mentioned factories, facades, the aggregator and the mediator component contain Hot Spots. In terms of connection to the tuple space and framework internal orchestration, Hot Spots are fully implemented but provide interfaces, which have to be extended problem specific: (a) a sensor façade and a respective physical object factory has to be implemented for every sensor; (b) an aggregator has to be implemented for every type of tangible artifact and (c) an application mediator and a corresponding application façade have to be implemented for every application to be connected to the framework.

4. Proof of Concept

Correct operation and stability of the framework was first tested using simulated sensor components. The practical feasibility was then shown in a real world scenario. During the simulation, long time tests were run to identify and debug e.g. malfunctions in the tuple space, memory problems or leaks and performance problems.

Simulation

The simulation environment implements a simple TUI scenario using two different types of sensors. The simulation of two different TUI elements (artifacts) led to the implementation of two aggregators. The gathered data was delivered to an application stub.

The scenario incorporates of two simulated sensor devices: a camera sensor and a magnetic field sensor device. Sensor input has been simulated using concurrent java threads. The following operations and/or combinations of operations have been examined: (1) Creation of visual and/or magnetic field objects; (2) Update visual and/or magnetic field objects; (3) Remove visual and/or magnetic field objects; (4) Identification of object types using information aggregation (either of the type "connector"
or "block"); (5) Delivery of sensed gathered data to a sample application stub; (6) Dynamic adding and removing of sensor components.

Besides the test for correct operation of the framework components, we have performed a 48 hours long time test using the simulation tool. The overall system remained stable for the duration of the test, even when sensors or the application were removed and added again. All the tests have been performed on a 2.0GHz AMD machine with 1GB of RAM and Windows XP.

An Actual Scenario of Use

The scenario in which the framework has been adopted implements an approach to externalize work task knowledge using tangible interfaces. The foundations of this approach application scenario have been developed in [21] and the overall implementation concept design is discussed in [22].

Figure 5 gives an overview of the system. It incorporates two cameras as sensor devices attached to two machines and a visualization application on the third one. The sensor components were built using the reactTVision-framework and its according markers [16]. A comprehensive description of the system is available in [23], photos and a video of the systems operation are available at [24].

Two types of phobs were deployed in the application scenario: modeling elements (blocks) representing different aspects of human work and connectors to describe the relationships between these elements.

During operation, the two sensor devices have been used to assign designators to modeling elements and to track element position respectively. For processing by the visualization application, the information delivered by each sensor has to be merged for every phob. This information is available asynchronously: designators are generally set first and remain stable over time; position becomes available as soon as the block is used for modeling and is updated continuously. The tuple-space based coordination mechanism is able to handle settings with asynchronous information sources by design without further configuration effort.

The framework proved to successfully orchestrated the given configuration and allowed real-time, synchronous tracking of physical artifacts. Performance issues while operating one camera on the same machine as the application were easily overcome. Using the distributed architecture of the framework, we added a dedicated machine for sensor operation. In the next stage of the evaluation, we aim at dynamic extension of the tracked workbench area at runtime by attaching further sensor devices when necessary.

6. Conclusions

The presented framework offers a robust, flexible means to collect information about tangible artifacts using arbitrary sensor technologies. It provides tailored information to one or more applications and is able to manage distribution of sensors, infrastructure components and applications. A comprehensive documentation of design, implementation, evaluation and usage of the framework can be found in [9].

Our framework has been designed to support collaboration of actors in work settings. It encourages a tuned mix of multiple (even spatially distributed) interactive systems (which include both sensors and software).

All identified requirements could be met both, from a conceptual perspective, and at the implementation level. We were also able to demonstrate user benefits: The TUIpist-framework provides potential users with an easy yet powerful means of conceptualisation and implementation of tangible user interfaces rooted in foundations of previous approaches.

In future research, we will evaluate the framework’s technology base and its applicability in collaborative settings of different domains. The claim for universal applicability has still to be validated.

7. References


