Funken
Serial Protocol Toolkit for Interactive Prototyping

Alexander Stefas¹, Andrea Rossi², Oliver Tessmann³
¹,²,³ Technische Universität Darmstadt
¹,²,³ {stefas|rossi|tessmann}@dg.tu-darmstadt.de

In order to offer a novel approach towards the development of interactive projects in architecture and design, as well as their tight integration in existing CAAD toolchains, this paper presents Funken, an open-source toolkit that handles serial communication for microcontrollers, aimed at simplifying the integration process between CAAD tools and interactive devices, and allowing fast implementation of human-readable user-specific communication protocols on the fly. Funken's details and implementation are presented, as well as custom-developed interfaces to Grasshopper, NodeJS and Processing. Funken is designed for building interfaces that allow users to implement their own custom defined logic, without imposing pre-determined behaviors. Within teaching, it allows to encapsulate complexity of microcontroller programming, while still allowing to implement complex behaviors through simple interfaces. The possibility of integrating Funken into a variety of CAD and media design frameworks offers the possibility of adding interactive functionality to a variety of projects.

Keywords: Serial Communication, Interactive Prototyping, Arduino, Physical Computing

INTRODUCTION
Today's built environment is constantly put under pressure by the increased pace of change needed to adapt to shifts in requirements of architectural artefacts (Achten, 2011). In response to demands of adaptivity and responsiveness for buildings, several researches have looked at possibilities of integrating interactive design and the use of microcontrollers, such as the Arduino tools ecosystem, into built projects (Anshuman, 2005), interactive installations (Ahlquist, 2015), as well as in educational curricula (Abdelmohsen and Massoud, 2015). Despite this expansion, current models of programming for responsiveness struggle to adapt to the specific needs of architectural objects, such as the ability to program aggregated behaviours without need for individual unit programming (Beal et al., 2012), as well as their integration into CAD platforms and algorithmic design tools.

In order to offer a novel approach towards the development of interactive projects in architecture and design, as well as their tight integration in existing CAAD toolchains, this paper presents Funken, an open-source toolkit that handles serial communication for microcontrollers, aimed at simplifying the integration process between CAAD tools and inter-
active devices, and allowing fast implementation of human-readable user-specific communication protocols on the fly.

BACKGROUND AND RELATED WORK
When it comes to the development of digital tools expanding the technological possibilities of CAAD environments, it is crucial to be able to adapt or develop these tools further, as well as allowing the user to do the same (Terzidis and Jungclaus, 2004). Indeed, many of today’s plug-ins in the vast ecosystem of computational design emerged from specific and narrow research questions, and hence their adaptability and reusability in other contexts is strictly dependent on the possibility of adapting these tools by accessing their source code. For this reason, during the development of Funken, attention has been paid to two aspects: from one side, the development of an open tool balancing ease of use and adaptability, and from the other, the possibility of interfacing the ecosystem to the widest possible variety of hardware projects and software tools.

Open-Source Thinking in Design and Prototyping
As described by Raymond (1999), two main models exist for the development of software: from one side, the “cathedral” offers a model where solutions are sold or provided as closed black boxes, and users are allowed to use the tool, but not understand its underlying functionality; on the contrary, the “bazaar” appears as a model where more fragmented solutions for problems must be orchestrated in the right way to solve a bigger problem. It is important to stress that, due to the contingent nature of tasks in design and of the use of programming within it (Kaijima and Michalatos, 2008), in most cases it becomes necessary to alter parts of tools and their source code in order to perform specific tasks. For this reason, the need for the code to be available under an open-source license, as well as the development of tools as open toolkits (Mackey and Sadeghipour, 2017) is a much-needed characteristic.

It is possible to notice that the idea of Open Source struggles to take root within research and practices related to the built environment. Indeed, due to economic and practical factors, most models seem to be based around “cathedral” business models that need to hide information or obfuscate pieces of software in order to maintain their authority, even though the tool or software is freely available. When it comes to the realm of physical computing and interactive prototyping, the continuous release of new hardware and specifications challenges the “cathedral” to either meet the new requirements or simply ignore new developments. Conversely, in a “bazaar” ecosystem, the numbers of solutions for problems will highly differ in quality and quantity, but the openness of the tools and their interoperability would allow to easily adapt existing solutions to novel developments (Von Hippel, 2001).

Electronic Prototyping Toolkits in CAAD
As already noted, current models of programming for responsiveness are hardly adapted to the needs of architectural design, as well as lack integration within existing CAAD packages. An interesting example in this direction is Firefly (Payne and Johnson, 2013), a Grasshopper plugin which offers possibility of controlling directly Arduino microprocessors from the program interface, and link sensor readings to geometric behaviors. Despite these abilities, the Firefly ecosystem is geared towards relatively simple interactive systems and supports few of the Arduino compatible boards. This last factor is particularly limiting when thinking of large scale distributed systems, where the ability to use smaller and cheaper boards than standard Arduino is relevant. Additionally, Firefly is not provided as open-source tool, and this limits the possibility of extending the language for non-standard applications or use within other design toolkit. On the Arduino side, Firefly relies on the Firefly Firmata which is a derivative of the Standard Firmata used in projects that rely on communicating via the serial port with a microcontroller. Such Firmata adds complexity and limitations to Arduino develop-
ment, as it is not microcontroller independent and uses a non-human-readable protocol.

Against sectoral approaches towards electronics engineering knowledge and aiming at allowing creative and dynamic interaction with electronic systems, different recent projects looked at modularization of component as valid strategy for increasing accessibility of such technologies to a wider audience. LittleBit (Bdeir, 2009) is a construction kit based on modular electronic elements, connected via magnetic pins to create interactive objects. The combination of magnets polarity and color coding of the units allows unskilled users to easily create functional electronic prototypes by combining few basic modules. Similarly, Cubelets (Grossand and Veicht, 2013) utilizes cubic modules with magnetic connection to create circuits and simple robotic systems. TinkerKit (Bellucci et al., 2014) provides a suite of integrated hardware and software nodes, with standardised connectors. Conversely, in order to avoid limitations or relying on a single hardware platform, Funken does not try to provide hard- and software solutions for common physical computing tasks. Rather, it aims at simplifying communication within a distributed hard- and software system that require the exchange of information like sensor data or messages that inform about application states or custom defined process events. The exchange of information is done in human readable form and does not require the use of binary or hexadecimal calculations. The exchange of simple strings provides the possibility to self-explain their purpose in a software architecture. This is especially valuable when used in non-expert environments like architectural studies or industrial design. Parts of the application logic could be externalized in the disciplines own toolset like Grasshopper.

METHODS
Funken is a toolkit that handles serial communication for microcontrollers. It has been optimized for the Arduino platform (Mellis et al., 2007) and consists of a library to include in any hardware project that uses Arduino and needs to communicate with surrounding infrastructure, such as other hard- or software components. It simplifies the communication process and makes it easy to implement user-specific protocols on the fly. The proposed workflow consists of a core Funken library for the Arduino ecosystem, as well as interfacing solutions for different creative coding and CAAD packages (Rhino/Grasshopper, Processing, NodeJS). Funken provides methods to build application logic and could be used to remote control a finite state machine in a microcontroller program, stream sensor data or build robotic end effectors or custom tools.

Serial Protocol Toolkit
The aim of Funken is to easily enable event-based functionality for microcontrollers, an approach that is best comparable to event-based programming, where an event object is associated with an event listening method (Figure 1). The event object is passed to the listening method and the method body is executed when a certain event is triggered (Faison, 2011).

This process is comparable to the definition of specific keywords to encode more complex information in a compact manner. By defining specific call-back
methods for events in Funken, and linking those to specific keywords, it becomes possible to build interactive systems where simple calls trigger complex software behaviour, hence allowing to encapsulate complexity of microcontroller programming into simpler macro-behaviours that are exposed to the user. This makes it possible to build custom human-readable serial protocols for Arduino, as well as to easily implement macro style behaviour, such as using “AnimationStart” as a keyword to trigger the execution of a custom set of methods (e.g., a specific pattern of LEDs blinks).

It is important to notice that, when communicating via a hardware connection, software processes are incomparably faster than their underlying hardware infrastructure. Due to this, it is only possible to send as much information as the recipient can process. Within Funken, serial messages, in the most common case of an Arduino UNO, are written to a 64 Byte FiFo Buffer (Wilenz, 1998). The written bytes can be taken from that buffer and processed directly on the board (Figure 2).

Every internal main loop execution now reads as many bytes from the input buffer as possible and decides on a byte by byte logic if a message is complete. A message is complete when the ASCII EOL character is read from the input buffer. A common practice when handling input via serial port. Funken uses 32 Bytes as an internal buffer for storing the incoming bytes. The maximum size of one message is half of the size of the internal Funken buffer, 16 Bytes.

When a message is identified Funken looks up the first characters in the message up until the first SPACE token and searches for these chars in array containing all the commands we registered in the beginning of our program. This is typically done in Arduino’s own setup method. An identified message finally executes the associated method with the complete message as a char array as argument. This way we could register call-backs and send messages that in the internal handling of that char array mimic the behaviour of arguments for that call-back.

**Interactive Prototyping Workflow**

A typical Funken implementation requires a small number of Arduino commands to be defined. After setting up Funken in the Arduino IDE it is possible to call the method “listenTo”, which links a text tokenizer (for example “DW”) to a specific call-back method (for example “funkenDigitalWrite”). This way Funken is informed that a message via the serialport that starts with “DW” should trigger the execution of the predefined method “funkenDigitalWrite”. This association is stored via “listenTo” in array of a custom struct which contains the message as a string and the method that should be executed. An example implementation of a call-back method is as follows:

```cpp
void funkenDigitalWrite(char *c) {
    char *token = strtok_r(c, " ", &p);
    char *pin = strtok_r(NULL, " ", &p);
    char *state = strtok_r(NULL, " ", &p);
    int pinint = atoi(pin);
    int stateint = atoi(state);
    byte debug = 0;
    if(pin != NULL && state != NULL) {
        debug = 1;
        digitalWrite(pinint, stateint);
    }
}
```

When an incoming serial message in the form “DW 13 1” is received by Funken, this triggers the execution of the method and switches pin 13 to high, for example lighting an LED connected to the selected pin. In this way it is possible to define very basic communication protocol for the Arduino application under development.

Through such workflow, Funken allows users to completely remote control an Arduino by remapping custom Serial commands like “DW” (abbreviation for digitalWrite) to Arduino built-in commands like digitalWrite(pin, value). This includes remapping pins as inputs or outputs on the fly as needed, a valuable possibility when working with small boards with limited number of I/O pins. Registered callback methods could also execute a set of standard Arduino functions. We send for example “ANIMATIONSTART” as a
serial message and execute a complex blinking pattern for some LED’s that are part of the application.

Funken can be easily extended with custom functionality without need for editing the core library. At the moment of writing this paper, a system of basic implementations is provided, serving as best practices for developing communication protocols and implementing use of other external libraries.

**Design Tools Interfaces**

It is important to understand that serial messages can come from every software layer that is able to connect to a serial port. Because serial communication is one of the most basic interfacing processes between software and hardware layers, methods for it are available in most programming languages and coding frameworks. Additionally, due to the simplicity of the communication infrastructure, its implementation is relatively low in complexity.

At the moment of writing the paper, interfaces for Grasshopper algorithmic modeling plug-in, for the Processing coding toolkit and for NodeJS have been developed and tested. More interfaces to other common coding and CAAD frameworks are under development (see Outlook section).

**Grasshopper.** The Grasshopper interface offers the possibility of fully remote-controlling any Arduino-compatible microcontroller through simple components. Through this process, it is possible to easily link different controller instances and pins to specific geometric and/or simulation-related values, computed using the variety of tools available in the Grasshopper ecosystem. The components allow to quickly convert the provided values into custom encoded commands in Funken, which are then streamed to the microcontrollers via serial communication, hence triggering the execution of the defined behaviors (Figure 3).

At the current stage, the plug-in allows to create listeners for each serial port available, through
which serial commands can be read and sent. This allows to listen to an unlimited number of Arduino-compatible boards connected to the same computer, hence allowing to build complex interactive systems with simple and small boards. Additionally, each board can be assigned a custom ID, in order to recognize it and send commands to the right board within the same Grasshopper definition. The basic set of components allows to mirror the basic functionality of Arduino code: pinMode, digitalWrite, digitalRead, analogWrite, analogRead. Additionally, it provides a component to send custom commands, which can be used to trigger custom Funken-defined callbacks, such as complex animations or readings from sensors with more complex interfaces than basic Arduino read methods.

The plug-in is written in Python, taking advantage of the PySerial serial communication library, an open-source implementation for serial communication within Python.

Other Interfaces. Due to the reliance of Funken on serial communication, one of the most common interfacing methods for hardware, it is possible to easily develop communication interfaces and plug-ins for various soft- or hardware frameworks used in CAD and media-art workflows. As example, NodeJS (Tilkov and Vinoski, 2010) was used to build a Web-server that renders a HTML Page with integrated WebSocket functionality. Via this socket a chat of connected users is possible. A special formatting of messages doesn’t send messages to other connected users but routes these messages to a serial connection on the server and control the connected Arduino with Funken. This allows to remote control Arduino devices via the internet, with a minimal amount of coding knowledge required to implement the system. Similarly, a basic Processing (Reas and Fry, 2006) implementation has been provided, allowing to use the accessible coding environment to build custom graphical interfaces to control electronic boards, without need to worry on how to implement communication protocols.

APPLICATIONS
The openness and ease of implementation of Funken allow its application in a variety of cases within existing design and fabrication workflows, as well as within CAD education courses. At the moment of writing the paper, Funken is still under development, but has been already tested as teaching aid in physical computing courses for architecture students, as well as implemented in robotic fabrication processes and in interactive projects.

Electronic Prototyping for Interactive Installations
The most direct application of Funken is the control of electronic devices to generate interactive prototypes and structures. Within such context, Funken simplifies the integration between Arduino-
controlled devices and other software and/or hard-
ware components, allowing to quickly orchestrate
them and control the outcome in response to specific
design or performance requirements.

For a modular robotic installation designed
within the authors’ department, Funken was used to
synchronize robotic arm’s motion with a DMX net-
work (Sid, 2001) of lights, allowing to control in real-
time the changes in light color (Figure 5). This has
been possible by translating the DMX protocol, one
of the standards within the lighting design world, to
a series of Funken call-backs, which could be fed the
id of the desired lamp and the RGB channels for the
desired light color. Similarly, for a robotic bar instal-
lation, depicted in Figure 4, Funken has been used
to allow users to interact with a beer-serving robot,
by using a button to request a next beer, as well as
to orchestrate a series of sensors used to control the
whole process (beer location tracking via distance
sensors, openable beer availability via light sensor).

**Interactive Processes for Robotic Fabrica-
tion**

Within the prototyping of custom robotic workflows
for digital fabrication and assembly, Funken offers a
fast way to define communication between robotic
control programs and external devices, through an
Arduino. This is an established process within ar-
chitectural research of robotic fabrication (Braumann
and Brell-Cokcan, 2012). However, most solution rely
on custom software components developed specif-
ically for each project, and a framework to quickly
implement interactions is still lacking. Funken offer
the possibility to be integrated with fast and low-cost
prototyping technologies (FDM 3D printing, laser-
cutting) to quickly design, manufacture and control
custom end-effectors. Additionally, work is being
directed at implementing custom Funken processes
to interact with the open-source plug-in Robots, de-
developed by Vicente Soler, for robotic programming,
in order to directly integrate microcontrollers ac-
tions and robotic motion into a single process. Cur-
rently, prototypes for an Arduino-controlled electro-
magnetic gripper and for a laser-bridge to monitor
assembly processes have been implemented.

**Electronic Prototyping in CAAD Education**

As a teaching aid to introduce students to interac-
tion design, Funken allows to present an easy-to-use
interface in a CAD software of choice (in our case,
Grasshopper), and to explain basic electronics pro-
gramming concepts without need to directly interact
with microcontrollers programming languages. The
level of simplification can be easily customized by
the course leader, by encapsulating complex behaviors
into a single Funken callback. Through this, students
can be exposed either to the same exact program-
ing logic used in microcontrollers programming
(digital read and write, pin mode definition, delay,
etc.), or could be provided with high-level abstract
control functions to trigger specific behaviors (eg.
LEDs animations or performing readings from ad-
vanced sensors, requiring custom interfacing code).

Within a seminar taught by the authors, Funken
has been used as interface for a variety of project in-
volving intercommunication between Grasshopper
and Arduino-based interactive devices. One exem-
plary project used Funken to link Grasshopper ge-
ometry generation with a violin player, by tracking
the position and pressure of the bow over the strings
with a distance and a pressure sensor and stream-
ing the recorded data in real-time to a custom-built
Grasshopper interface. Another project used Funken
to orchestrate a variety of sensors and actuators, al-
lowing to build a Grasshopper-controlled physical replica of a retro videogame, while at the same time simulate the whole gameplay within the Grasshopper interface itself (Figures 6-7).

**Modular Electronics**

Within the realm of interactive design, Funken is being coupled with current research in modular design and discrete fabrication (Rossi and Tessmann, 2017), allowing to embed custom functionality into individual modules, to be assembled into structures, where different modules can communicate and perform custom behaviors. This has been possible by integrating Funken with developed methods for discrete design, through the Wasp plug-in developed by the authors, and by modularizing both design and control commands generation (Figure 8). This makes it possible to link geometric design features (e.g. distance from control geometries, position in the aggregation, etc.) to specific aggregated behaviors, which are then routed through Funken to the specific control boards of each individual module.
CONCLUSION

Discussion of Results

Funken is designed for building systems that allow users to implement their own custom defined logic, without imposing pre-determined behaviors. Within teaching, it allows to encapsulate complexity of microcontroller programming, while still allowing to implement complex behaviors through simple interfaces. The possibility of integrating Funken into a variety of CAD and media design frameworks offers the possibility of adding interactive functionality to a variety of projects. With the aim of encouraging tinkering and experimental prototyping, Funken is explicitly readable from the outside and has no built-in safety mechanisms. This allows it to be open and flexible enough to allow users to quickly sketch a system’s behavior and implement it on Arduino-compatible microcontrollers.

The toolkit has been tested in a teaching environment, as well as in some initial interactive prototypes and robotic processes. It is planned to be released as open-source project in the summer of 2018.
Outlook
The next planned step is the complete documentation of the framework, as well as the provision of exemplary implementations for a variety of devices and interfaces. Additionally, the source code is currently being expanded to handle I2C communication, with the aim of offering the possibility of building larger networks of more simple devices, without the need of a microcontroller for each component in the system. Lastly, development of interfaces for other frameworks, such as OpenFrameworks, Raspberry Pi, VVVV, MaxMSP, ROS, MatLab/Simulink, and others is also possible. While the authors are planning to implement some of these in the future, the hope is that releasing the framework with proper documentation will encourage users to develop their own interfaces and contribute them to the project.

ACKNOWLEDGEMENTS
Funken is developed and maintained by A. Stefas, while the Grasshopper interface is developed by A. Rossi. Both have been developed as part of the authors' research at the DDU Digital Design Unit at TU Darmstadt. The toolkit was tested during seminars taught by the authors at TU Darmstadt. The student project documented in the paper have been carried out by P. Salas, P. Morales, R. Winkler. The Luminale modular robotic installation was developed by DDU, led by O. Tessmann and B. Wisranek. The Robot Bar project was developed by the authors, with support from T. Gruner and S. Bisswanger. Initial development of Funken was inspired by the SerialCommand library by S. Cogswell.

REFERENCES
Bdeir, A 2009 ‘Electronics as material: littleBits’, Proceedings of the 3rd International Conference on Tangible and Embedded Interaction
Beal, J, Dulman, S, Usbeck, K, Viroly, M and Correll, N 2012, Organizing the aggregate: Languages for spatial computing, CoRR
Faison, T 2011, Event-Based Programming, APress
Grossand, MD and Veitch, C 2013, ‘Beyond top down: Designing with cubelets’, Tecnologias, Sociedade e Conhecimento, 1 (1), pp. 150-164
Kaijima, S and Michalatos, P 2008 ‘Simplexity, the programming craft and architecture production’, First International Conference on Critical Digital: What Matters(s)?
Mackey, C and Sadeghipour Roudsari, M 2017 ‘The Tool(s) vs. The Toolkit’, Proceedings of the Design Modeling Symposium, Paris
Sid, A, MAF Tech Corp 2001, Addressable light dimmer and addressing system, U.S. Patent 6,175,201
Terzidis, K and Jungclaus, J 2004, Predicting the Future: Open Source CAAD, Harvard University Publication